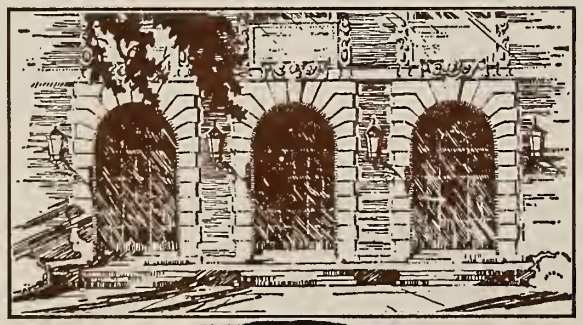


LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
I26r
no. 541-546
cop. 2





Digitized by the Internet Archive
in 2013

<http://archive.org/details/automaticsolutio546lars>

10.84
26n
0.546
of.2

UIUCDCS-R-72-546

Math

COO-1469-0212

AUTOMATIC SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS

by

Leonard Andrew Larsen

October 1972

JAN 29 1973

THE LIBRARY OF THE

DEC 1 1972

UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-72-546

AUTOMATIC SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS*

by

Leonard Andrew Larsen

October 1972

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

* Supported in part by the Department of Computer Science and in part by the Atomic Energy Commission under contract US AEC AT(11-1)1469 and submitted in partial fulfillment of the requirements of the Graduate College for the degree of Doctor of Philosophy in Computer Science, October 1972.

ACKNOWLEDGMENT

I am indebted to Professor C. W. Gear for advising this thesis;
I appreciate his suggestions and guidance.

I am also grateful to the Atomic Energy Commission for supplying
the computer time needed to complete this research.

TABLE OF CONTENTS

	<u>Page</u>
1. INTRODUCTION.	1
2. PROCEDURES.	4
2.1 <u>Spatial Discretization</u>	4
2.2 <u>Error Estimates</u>	18
2.3 <u>Efficiency</u>	30
2.4 <u>Initial Partition</u>	35
2.5 <u>Changes in Integration</u>	38
2.6 <u>The User's Equations</u>	40
3. PROGRAMS.	43
3.1 <u>PARDIF</u>	43
3.2 <u>XSTP</u>	46
3.3 <u>DERIV</u>	48
3.4 <u>DIFSUB</u>	52
3.5 <u>DECOMP and SOLVE</u>	54
3.6 <u>DIFFUN</u>	54
3.7 <u>DIVERR</u>	55
4. RESULTS	56
LIST OF REFERENCES.	74
APPENDICES	
A. THE SYSTEM OF PROGRAMS.	75
B. EXAMPLE 1	119
C. EXAMPLE 2	122
D. EXAMPLE 3	125
VITA.	129

1. INTRODUCTION

Define an automatic program as one that can

1. take as input members of some broad class of problems,
2. try to control errors within specified limits,
3. try to minimize the work required for the solution.

The problem of the present paper is to look for an automatic method that can be applied to some subset of partial differential equations. The primary value of a program involving an automatic method would be to provide the person who needs only a few runs with a particular type of equation the opportunity to find a solution, within acceptable tolerances, without having to write and debug a specialized program. In this environment, the program will have no information about the solution, or how errors might accumulate. While maintaining sufficiently small errors, an attempt needs to be made to keep the cost of such a solution as low as possible. To accomplish these goals, decisions need to be made during the integration for balancing the two conditions above based on some sort of error estimate and the user's error tolerance.

In the present study, the subclass of partial differential equations was chosen to be the set of second order mixed initial boundary value problems in two variables (x and t) of the form $F(u, u_x, u_{xx}, x, t) = u_t$. The general method used involves integrating along lines which are parallel to the time axis and which pass through a finite set of points on the spatial axis. To obtain an idea of this method, consider the equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

with initial condition

$$u(x,0) = \sin(\pi x) \quad x \text{ in } [0,1]$$

and boundary conditions

$$u(0,t) = u(1,t) = 0$$

To begin, select a partition

$$P = \{0 = x_0, x_1, x_2, \dots, x_{n-1}, x_n = 1\}$$

and then define a set of functions $u_i(t)$ such that

$$u_i(0) = u(x_i, 0) \quad i = 1, 2, \dots, n.$$

By the use of some interpolating polynomial, a discrete analog is obtained for the spatial derivative, which will be represented by

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x_i} \cong A_n(u_i)$$

Using this analog, we obtain the discrete system

$$\frac{du_i}{dt} = A_n(u_i) \quad i = 1, \dots, n$$

This system differs from the system

$$\left. \frac{\partial u(x,t)}{\partial t} \right|_{x_i} = \left. \frac{\partial^2 u(x,t)}{\partial x^2} \right|_{x_i}$$

which is obtained from the true solutions by an amount $\delta(i,t)$.

The analog $A_n(u_i)$ and the partition P are chosen in an attempt to keep $u_i(t) - u(x_i, t)$ within the user's desired tolerance.

Once the partition and analog are chosen, one then integrates the system in the time direction along the lines implied by P from $t = t_0$ to $t = t_1$. If at any time during this integration it appears that the error

is too large, the step is repeated with the possibility of changing either $A_n(u_i)$ or P . A change in the partition would mean continuing the integration along a new set of lines.

There is also the possibility that the error is much smaller than is required. Again, modifications in $A_n(u_i)$ or P might be attempted so that the work of obtaining the desired error is minimized. These decisions will need to be made on the basis of some estimate of the error.

The basis for the integration of the ordinary differential equation system above is a program (DIFSUB) developed by Gear [1] for automatically solving systems of stiff ordinary differential equations. As an automatic program, it attempts to keep the errors introduced in integrating in the t direction within a given tolerance.

In integrating along the possibly changing set of lines, there are thus three ways of trying to keep the errors in the solution at a desired value. First, the partition can be changed. Second, the nature of the analog used to discretize the system can be altered. And third, the amount of the error parameter given to the integration routine can be modified.

In order to achieve an automatic program for the entire system, several items will need to be considered:

1. the forms of the analogs to be used,
2. the methods of obtaining error estimates,
3. the methods of obtaining and modifying partitions,
4. the implementation of the user's equations in an easily usable form.

2. PROCEDURES

2.1 Spatial Discretization

There are several problems to be considered in looking for a spatial analog. These include the number of equations required, the ease of changing the partition, and the situation at the boundaries. Three methods were initially considered:

1. Lagrangian methods,
2. Hermite type methods,
3. Linear transformations of the Lagrangian methods.

These will then be used on equations of the form

$$\frac{\partial u}{\partial t} = f(u, u_x, u_{xx}, x, t)$$

The equations are then put into a discrete form by replacing u by a set of functions $u_i(t)$ and replacing the right side by an analog $A_n(u_i)$. This then gives the system of equations

$$\frac{du_i}{dt} = A_n(u_i) \quad (1)$$

In the case of the Lagrangian polynomials, there are several possibilities. First, there is the class of polynomials using equidistant points. If the number of points (n) in the interpolating polynomial is odd, then it is possible to take advantage of symmetry properties to eliminate one of the terms in the error. Given the approximation

$$y(x) = \sum_{i=1}^n l_i(x) f(a_i) \quad (2)$$

to a function $f(x)$ for interpolating points $a_1, \dots, a_n$, the error term is given by

$$\frac{p_n(x)}{n!} f^{(n)}(\eta)$$

where $p_n(x) = (x-a_1)\dots(x-a_n)$ and η is some point in the interval containing these points. Taking the derivative of the interpolating polynomial, an approximation is obtained for $f'(x)$.

$$y'(x) = \sum_{i=1}^n \ell_i'(x) f(a_i) \quad (3)$$

To obtain ideas about the error, take the derivative of the error term of (2) (see 2.2 for discussion and development). For equation (3), the error term would be given by

$$\frac{p_n'(x)}{n!} f^{(n)}(\eta_1) + \frac{p_n(x)}{(n+1)!} f^{(n+1)}(\eta_2) \quad (4)$$

Since $p_n(x) = 0$ at tabular points, where all such evaluations will be made, (4) reduces to

$$\frac{p_n'(x)}{n!} f^{(n)}(\eta_1) \quad (5)$$

If this process is repeated, the error for the second derivative is

$$\frac{p_n''(x)}{n!} f^{(n)}(\eta_1) + \frac{2p_n'(x)}{(n+1)!} f^{(n+1)}(\eta_2) + \frac{2p_n(x)}{(n+2)!} f^{(n+2)}(\eta_3) \quad (6)$$

If the interpolating polynomial uses an odd number of points, and if the derivative is estimated for the center point, then the first term is zero.

Consider

$$p_n''(x) = \sum_{\substack{i=1 \\ j \neq i}}^n \sum_{j=1}^n \frac{p_n(x)}{(x-a_i)(x-a_j)}$$

Since evaluation is at the center point, $x - \frac{a_{n+1}}{2} = 0$, and all terms containing this factor will vanish. As a result, $p_n''(x)$ can be rewritten as (where the subscript $n1$ is $\frac{n+1}{2}$)

$$p_n''(x) = \sum_{\substack{i=1 \\ i \neq n_l}}^n \frac{2 \cdot p_n(x)}{(x-a_{n_l})(x-a_i)} \quad (7)$$

Further, since the points are symmetrically located around a_{n_l}

$(a_{n_l} - a_{n_l} + i) = (a_{n_l} - a_{n_l} - i)$. Noting this symmetry, (7) can be written

$$p_n''(x) = \sum_{i=1}^{a_{n_l}} \left(\frac{2 \cdot p_n(x)}{(x-a_{n_l})(x-a_{n_l} + i)} + \frac{2 \cdot p_n(x)}{(x-a_{n_l})(x-a_{n_l} - i)} \right) = 0$$

Since the last term of (6) is also zero, the expression reduces to

$$\frac{2p_n'(x)}{(n+1)!} f^{(n+1)}(x) \quad (8)$$

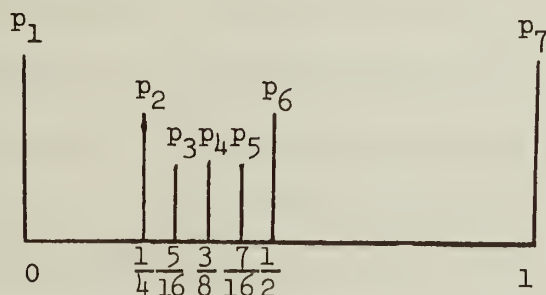
Since $p_n(x)$ is a function of the spacing h , the order could be defined as the lowest power of h contained in the error. In (5) and (8) these orders are the same, so that the truncation errors should react similarly to changes of h . In the event the symmetry is lost, the order associated with the second derivative is reduced by one, and $p_n''(x)$ is no longer zero. The error term for the second derivative would probably then be the determining factor in selection of an h .

One of the main difficulties with this class of methods results from the equidistant spacing. If the solution is "non-smooth", the spacing for the partition will have to be set so as to reduce the error at the worst point. This could lead to a much larger number of equations than needed, for in the smoother regions the error would be expected to be much less than is required. Ideally, one is aiming to have partition points

such that the total error introduced at each is at the level desired by the user.

In an attempt to keep the error closer to the user's request, one could divide the interval into subintervals with spacings that are locally equidistant. If the analog uses five or more points, such subdivisions have problems near the end of each subinterval. The derivative values near these subinterval end points would either have to use a nonsymmetric form or a point from an adjacent interval with possibly a different step size.

When using a three-point analog, these problems are reduced. Each analog could use spacings that are multiples of some h . In order to circumvent the problem of the end point, one could consider it as the center point of another subinterval overlapping those on either side. For example, given the partition



and an $h = \frac{1}{16}$, the analogs could be set up in the following way

<u>derivative at</u>	<u>points used</u>
P_2	P_1, P_2, P_6
P_3	P_2, P_3, P_4
P_4	P_3, P_4, P_5
P_5	P_4, P_5, P_6
P_6	P_1, P_6, P_7

In practice, enough points would be added so that several of the sub-intervals would not be contained within another interval.

Another way to try to keep all errors somewhat close to the user's request would be to use unequal spacings throughout the interval. There are several disadvantages to this approach. First, the order of the error term would be lower than those associated with symmetric-equispaced ones. Second, there exists the problem of calculating the derivatives. With equidistant points, a set dependence exists between values of the function and the derivative analog at any partition point. In the case of unequal spacings, either the analog would have to be calculated each time it was needed or the dependence of $A_n(u_i)$ on each point would have to be calculated and then stored for a given partition. Third, the initial partition would be somewhat more difficult to develop because of the freedom of changing the spacings between the points. As a result, it could happen that the spacings between any two pairs of adjacent points could be greatly different. In this case, it would be desirable to go back and change other points so as to avoid widely differing spacings for neighboring points.

A second class of analogs involves those methods that use more than just the functional value at each point. The primary subclass would be the Hermite interpolating polynomials. With polynomials of this form, it is necessary to save the first derivative at each point as well as the functional value. In order to develop this additional value at each time step, it would be necessary to solve an additional equation at each point. Such an equation can be found by taking the partial derivative with respect to the space variable of both sides of the given equation

$$u_t = f(u, u_x, u_{xx}, x, t) \quad (9)$$

This gives

$$\frac{\partial}{\partial x} u_t = \frac{\partial}{\partial x} f(u, u_x, u_{xx}, x, t) \quad (10)$$

Since the solution u is assumed to be continuous and to have as many continuous partials as may be required, the order of differentiation is changed to give

$$\frac{\partial}{\partial t} u_x = \frac{\partial}{\partial x} f(u, u_x, u_{xx}, x, t) \quad (11)$$

At this point, u_x is replaced by a new variable v_i and the right side of (11) is replaced by a discrete analog $B_n(u_i, v_i)$, giving

$$\frac{\partial}{\partial t} v_i = B_n(u_i, v_i) \quad (12)$$

The solution v_i of this addition equation is not expected to be ill-behaved provided the solution is continuous. Consider the Taylor series expansion of u

$$u(x, t) = u(x_0, t_0) + u_x(x_0, t_0)(x - x_0) + u_t(x_0, t_0)(t - t_0) + \dots$$

The presence of u_x in this expansion indicates that any unusual behavior that might be reflected in u_x would be present in u . As a result, if the original system of equations (9) can be solved, then it is expected that the combined system of (9) and (11) can also be solved.

Just as in the Lagrangian case, there is an advantage in using the symmetry properties to eliminate terms in the expression for the error. Further, all of the problems associated with the Lagrangian polynomials--when more than three points are involved--are compounded by the fact that the first derivative must be calculated and saved for each point. A new problem exists in that additional values are needed at the boundary. In

the case of the original equations, the boundary conditions provide additional equations that can be solved with the differential equations. With v_1 and v_n at the boundaries, it will be necessary to develop other equations, or at least provide values for these variables.

Another possible member of this second class of interpolating polynomials would be those that match first and second derivatives as well as the functional values at the interpolating points. The use of such polynomials (trimate) and the problems involved will be very similar to those encountered in the Hermite case. Here, two additional equations will need to be supplied for each point. The first of these would be the same as the additional equation in the Hermite case. The second would be found by differentiating both sides of (11) with respect to x giving

$$\frac{\partial}{\partial x} \frac{\partial}{\partial t} u_x = \frac{\partial^2}{\partial x^2} f(u, u_x, x_{xx}, x, t)$$

Interchanging the order again, introducing an additional variable $w_i \equiv \frac{dv_i}{dx}$ and an analog $C_n(u_i, v_i, w_i)$, the discrete version becomes

$$\frac{dw_i}{dt} = C_n(u_i, v_i, w_i) \quad (13)$$

Another problem, that was also present in the Hermite but becomes more evident in the trimate polynomials, is the presence of higher ordered derivatives. The error term for the trimate will contain the ninth and tenth derivatives. Although accuracy will not be required, a reasonable approximation is desirable.

The one additional possibility that was considered for obtaining an analog was to use a linear transformation of a Lagrangian method which would save different quantities, but which would possibly save work. Suppose

and that for the second derivative is ($D = TCT^{-1}$)

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & \frac{1}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & -4 & -2 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & -2 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 2 & 0 & -4 & \frac{1}{2} & -4 & -2 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & -\frac{1}{2} & 0 & 0 \\ 2 & 0 & -4 & 0 & -4 & \frac{1}{2} & -4 & -2 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since the problems the procedures are meant to solve could involve both derivatives, the whole idea of using linear transformations does not appear to provide anything of interest.

As a result of the disadvantages of going to more than three points it was decided that three methods would be used. These were the Lagrangian, Hermite, and trimite polynomials using three points. This would provide methods of three different orders. Although the Hermite and trimite forms have additional problems, the flexibility of having the different ordered methods and the possibility of a fewer number of equations makes consideration of these methods worth undertaking.

If the polynomials for Lagrangian, Hermite, and trimite are called Y_L , Y_H , Y_T , respectively, the resulting formulas can be written

$$Y_L = \sum_{i=1}^n \ell_i(x) f(x_i)$$

$$Y_H = \sum_{i=1}^n (h_i(x) f(x_i) + \bar{h}_i(x) f'(x_i))$$

$$Y_T = \sum_{i=1}^n (t_i(x) f(x_i) + \bar{t}_i(x) f'(x_i) + \bar{\bar{t}}_i(x) f''(x_i))$$

where $t_i(x)$, $\bar{t}_i(x)$, and $\bar{\bar{t}}_i(x)$ are polynomials associated with the trimite method.

The development of t , $\bar{t}$, $\bar{\bar{t}}$ parallels the development of the Hermite polynomials.

Let

$$t_k = (a_{1k} x^2 + a_{2k} x + a_{3k}) \ell_k^3(x)$$

$$\bar{t}_k = (b_{1k} x^2 + a_{2k} x + a_{3k}) \ell_k^3(x)$$

$$\bar{\bar{t}}_k = (c_{1k} x^2 + c_{2k} x + c_{3k}) \ell_k^3(x)$$

There are three conditions for each of the equations

$$t_k(x_j) = \delta_{kj} \quad t'_k(x_j) = 0 \quad t''_k(x_j) = 0$$

$$\bar{t}_k(x_j) = 0 \quad \bar{t}'_k(x_j) = \delta_{kj} \quad \bar{t}''_k(x_j) = 0$$

$$\bar{\bar{t}}_k(x_j) = 0 \quad \bar{\bar{t}}'_k(x_j) = 0 \quad \bar{\bar{t}}''_k(x_j) = \delta_{kj}$$

Solving each row gives

$$a_{1k} = 6(\ell'_k(x_k))^2 - \frac{3}{2} \ell''_k(x_k)$$

$$a_{2k} = -(12x_k \ell'_k(x_k) + 3)\ell'_k(x_k) + 3x_k \ell''_k(x_k)$$

$$a_{3k} = 1 - \frac{3x_k^2}{2} \ell''_k(x_k) + (6x_k^2 \ell'_k(x_k) + 3x_k) \ell'_k(x_k)$$

$$b_{1k} = -3\ell'_k(x_k)$$

$$b_{2k} = 1 + 6x_k \ell'_k(x_k)$$

$$b_{3k} = -x_k(1 + 3x_k \ell'_k(x_k))$$

$$c_{1k} = \frac{1}{2}$$

$$c_{2k} = -x_k$$

$$c_{3k} = \frac{1}{2}x_k^2$$

Substituting into the original expressions and simplifying, the equations become

$$t_k(x) = [1 - \frac{3}{2}(x-x_k)^2 \ell''_k(x_k) + 6(x-x_k)^2 (\ell'_k(x_k))^2 - 3(x-x_k) \ell'_k(x_k)] \ell_k^3(x)$$

$$\bar{t}_k(x) = [(x-x_k) - 3\ell'_k(x_k)(x-x_k)^2] \ell_k^3(x)$$

$$\bar{\bar{t}}_k(x) = \frac{1}{2}(x-x_k)^2 \ell_k^3(x)$$

Using a proof by Wendroff [2], the error term is

$$\frac{p^3(x)}{n!} f^{(n)}(\eta)$$

where n is the number of values involved. Since a three-point formula for the derivative of the center point is being used, the coefficients for each of these interpolating polynomials can be determined.

When using the Lagrangian polynomials, the given equations are used, so the first and second derivatives are needed. These are

$$Y'_l = (f(x+h) - f(x-h))/2h$$

$$Y''_l = (f(x+h) - 2f(x) + f(x-h))/h^2$$

In the case of the Hermite, an additional equation was introduced that might require the third derivative. However, since the first derivative is saved for the partition point, no calculation is needed there. As a result, the derivatives needed are

$$Y''_h = \frac{2(f(x+h) - 2f(x) + f(x-h))}{h^2} - \frac{f'(x+h) - f'(x-h)}{2h}$$

$$Y'''_h = \frac{15(f(x+h) - f(x-h))}{2h^3} - \frac{3(f'(x+h) + 8f'(x) + f'(x-h))}{2h^2}$$

Similarly, the trimite forms require the third and fourth derivatives, but do not need the first or second. These are

$$Y'''_t = \frac{105(f(x+h) - f(x-h))}{8h^3} - \frac{\frac{33}{8}(f'(x+h) + f'(x-h)) + 18f'(x)}{h^2} + \frac{3(f''(x+h) - f''(x-h))}{8h}$$

$$Y''''_t = \frac{72(f(x+h) - 2f(x) + f(x-h))}{h^4} - \frac{39(f'(x+h) - f'(x-h))}{2h^3} + \frac{3(f''(x+h) - 24f'(x) + f''(x-h))}{2h^2}$$

In implementing the methods given above, consideration needs to be given to the method of storage. In the methods involving the generation of an initial partition, it would be possible to generate a natural order of points or in some order that is suggested by the method of generating the

points. If the initial partition is generated in something other than the natural sequence, it might be better to reorder. This question is answered by looking at the subroutine used to solve the system of differential equations. In this subprogram there are several places where the user's function is needed and the values of the partition points will be accessed. Each access will require that an order vector be used to obtain the positional value. There are also several other arrays in this subprogram. If these are stored in the order that the partition is stored, further time is consumed in determining that address. If these additional arrays are stored in an order different from that of the partition, then some problems will occur whenever the partition needs to be changed. For these reasons, it was decided to maintain the natural order, regardless of how the partition is generated.

2.2 Error Estimates

If the method is going to be automatic, a part of the procedure must be devoted to calculating some sort of estimate of the error and deciding some course of action based on the results. There are basically three areas that need to be discussed. Two of these involve being able to determine an estimate of errors introduced through spatial discretization, and also an estimate of errors incurred by solving the system of ordinary differential equations. The first type of error will be called d_x and the second d_t . The third area of discussion involves the need to combine these errors to produce a result that gives some measure of the overall error.

In equation (1), the right hand side, $A_n(u_i)$, can be written as

$$f(u_i, A_{n,x}(u_i), A_{n,xx}(u_i), x_i, t) \quad (14)$$

where $A_{n,x}(u_i)$ and $A_{n,xx}(u_i)$ are analogs of the first and second derivatives, respectively. Putting this expression in place of $A_n(x_i)$, (1)

$$\frac{du_i}{dt} = f(u_i, A_{n,x}(u_i), A_{n,xx}(u_i), x_i, t) \quad (15)$$

The analogs $A_{n,x}(u_i)$ and $A_{n,xx}(u_i)$ produce a truncation error $d_x(x_i, t)$. An estimate of $d_x(x_i, t)$ can be obtained by perturbing the right hand side of equation (15) by the error estimates of $A_{n,x}(u_i)$ and $A_{n,xx}(u_i)$ (e_1 and e_2 , respectively). This would give the expression

$$f(u_i, A_{n,x}(u_i) + e_1, A_{n,xx}(u_i) + e_2, x_i, t)$$

Subtracting the expression (14) from this, the result

$$d_x(x_i, t) = f(u_i, A_{n,x}(u_i), A_{n,xx}(u_i), x_i, t) - f(u_i, A_{n,x}(u_i) + e_1, A_{n,xx}(u_i) + e_2, x_i, t) \quad (16)$$

is obtained.

In order to be able to obtain this estimate, the values of e_1 and e_2 will have to be obtained. If the analogs involved were just those of the Lagrangian polynomials, then several approaches are available for obtaining the error terms e_1 and e_2 . Since the present approach is to use Hermite type polynomials as well, the approach of Wendroff [2] will be followed.

If $y(x)$ is any polynomial approximation to $f(x)$, then

$$f(x) = y(x) + P(x) G(x) \quad (17)$$

where $P(x)$, $G(x)$ is the error term, $G(x)$ is an unknown function,

$$P(x) = (x-a_1)^{m_1} (x-a_2)^{m_2} \dots (x-a_k)^{m_k}$$

and m_i is the number of values obtained at a_i .

In order to find the error estimate of $f^{(j)}(x)$ (the trimite method would require that $j=4$), take the j -th derivative of both sides of (17).

For the first four derivatives (assuming that G is continuous and has j continuous derivatives) the results obtained would be

$$\begin{aligned}
 f'(x) &= y'(x) + P'(x) G(x) + P(x) G'(x) \\
 f''(x) &= y''(x) + P''(x) G(x) + 2P'(x) G'(x) + P(x) G''(x) \\
 f'''(x) &= y'''(x) + P'''(x) G(x) + 3P''(x) G'(x) + 3P'(x) G''(x) + P(x) G'''(x) \\
 f^{(4)}(x) &= y^{(4)}(x) + P^{(4)}(x) G(x) + 4P'''(x) G'(x) + 6P''(x) G''(x) + 4P'(x) G'''(x) \\
 &\quad + P(x) G^{(4)}(x)
 \end{aligned} \tag{18}$$

The first problem is to determine the validity of the assumption that $G, G', G'', G''', G^{(4)}$ are actually continuous functions. To do this, for G , solve (17) for G giving

$$G(x) = \frac{f(x) - y(x)}{P(x)} \tag{19}$$

Since $f(x)$ is assumed continuous, $G(x)$ is continuous whenever $P(x) \neq 0$.

Consider now a point a_i , $P^{(j)}(a_i) = 0$ for $0 \leq j < m_i$ but $P^{(m_i)}(a_i) \neq 0$.

It is also noted that $[f(x) - y(x)] = 0$ for $0 \leq j < m_i$. Therefore, $G(a_i)$ is defined by

$$G(a_i) = \frac{f^{(m_i)}(a_i) - y^{(m_i)}(a_i)}{P^{(m_i)}(a_i)}$$

which by L'Hospital's rule shows the continuity of G .

To show that $G'(x)$ is continuous, Wendroff [2] takes the derivative of both sides of (19) giving

$$G'(x) = \frac{P(x)[f'(x) - y'(x)] - P'(x)[f(x) - y(x)]}{[P(x)]^2} \tag{20}$$

He then defines

$$h(x) = (x - a_i) \frac{P'(x)}{P(x)} = (x - a_i) \sum_{j=1}^k \frac{m_j}{(x - a_j)}$$

where k is the number of interpolating points. Substituting this into (20) and using L'Hospital's rule again, he arrives at the continuity of $G'(x)$

provided $f^{(m_i+1)}(x)$ is continuous for the maximum m_i .

The same approach can be used to show that each of G'' , G''' , G'''' is also continuous provided f has a sufficient number of continuous derivatives. Since the only major difference is that in each succeeding case the computations become much more involved, only the proof of the continuity of G'' will be shown.

To this end, take the derivative of both sides of (20) giving

$$G''(x) = \frac{2\left(\frac{P'(x)}{P(x)}\right)^2 (f(x)-y(x)) - \frac{P''(x)}{P(x)} (f(x)-y(x))}{P(x)} - \frac{2\left(\frac{P'(x)}{P(x)}\right) (f'(x)-y'(x)) + (F''(x)-y''(x))}{P(x)} \quad (21)$$

Again, the problem of continuity exists only at points where $P(x) = 0$, since otherwise all functions are continuous. So consider a point a_i where $P(x)$ has m_i zeros. Multiply the numerators and denominators of (21) by $(x-a_i)^2$ giving (let $f \equiv f(x)$, $y \equiv y(x)$, $P \equiv P(x)$)

$$G''(x) = \frac{2(x-a_i)^2 \left(\frac{P'}{P}\right)^2 (f-y) - (x-a_i)^2 \frac{P''}{P} (f-y) - 2 \cdot \frac{P'}{P} (f'-y')}{(x-a_i)^2 P} + \frac{(x-a_i)^2 (f''-y'')}{(x-a_i)^2 P} \quad (22)$$

Since the denominator has m_i+2 zeros at a_i and the (m_i+2) -nd derivative is nonzero, the limit of $G''(a_i)$ exists--provided the numerator

vanishes for the first m_i+1 derivatives. The continuity of $G'(a_i)$ and definition of the derivative then would allow the conclusion that $G''(a_i)$ is continuous. It is assumed that $f(x)$ has a sufficient number of continuous derivatives.

Define

$$h(x) = (x-a_i) \frac{P'}{P} = (x-a_i) \sum_{j=1}^k \frac{m_j}{(x-a_j)}$$

and

$$d(x) = (x-a_i)^2 \frac{P''}{P} = (x-a_i)^2 \left[\sum_{\substack{j,l=1 \\ j \neq l}}^k \frac{m_j m_l}{(x-a_j)(x-a_l)} + \sum_{j=1}^k \frac{m_j(m_j-1)}{(x-a_j)^2} \right] \quad (23)$$

It is noted that $h'(x)$ and $d'(x)$ are then given by

$$h'(x) = \sum_{\substack{j=1 \\ i \neq j}}^k \left[\frac{m_j}{(x-a_j)} - \frac{(x-a_i)m_j}{(x-a_j)^2} \right]$$

and

$$\begin{aligned} d'(x) = & \sum_{\substack{j,l=1 \\ j \neq l \neq i}}^k \left[\frac{2(x-a_j) m_j m_l}{(x-a_j)(x-a_l)} - \frac{2(x-a_i)^2 m_j m_l}{(x-a_j)^2 (x-a_l)} \right] \\ & + 2 \sum_{\substack{j=1 \\ i \neq j}}^k \left[\frac{m_i m_j}{(x-a_j)} - \frac{(x-a_i) m_i m_j}{(x-a_j)^2} \right] \\ & + \sum_{\substack{j=1 \\ i \neq j}}^k \left[\frac{2(x-a_i) m_j (m_j-1)}{(x-a_j)} - \frac{(x-a_i)^2 m_j (m_j-1)}{(x-a_j)^2} \right] \quad (24) \end{aligned}$$

If (23) and (24) are evaluated at a_i , they become

$$h(a_i) = m_i, \quad d(a_i) = m_i(m_i - 1),$$

$$h'(a_i) = \sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{(x-a_j)}, \quad d'(a_j) = 2m_i \sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{(x-a_j)}$$

Substituting these into (22), the numerator $G''(x)$ becomes

$$2h^2(x)(f-y) - d(x)(f-y) - 2(x-a_i)h(x)(f'-y') + (x-a_i)^2(f''-y'') \quad (25)$$

The derivatives of each of the four terms (call them A, B, C, and D) in (25) are now evaluated using Leibnitz' rule. This gives

$$1. \quad A \equiv 2 \frac{d^\alpha}{dx^\alpha} (h^2(x)(f-y)) = \sum_{r=0}^{\alpha} \binom{\alpha}{r} (h^2(x))^{(r)} (f-y)^{(\alpha-r)}$$

This term is now evaluated at a_i . Since $(f-y)$ has m_i zeros, A is identically zero for $0 \leq \alpha < m_i$

$$\text{for } \alpha = m_i, \quad A = 2 \cdot h^2(x)(f-y)^{(m_i)} = 2m_i^2(f-y)^{(m_i)}$$

$$\text{for } \alpha = m_{i+1},$$

$$\begin{aligned} A &= 2 \cdot h^2(x)(f-y)^{(m_i+1)} + 4h(x) \cdot h'(x)(f-y)^{(m_i)} \\ &= 2m_i^2(f-y)^{(m_i+1)} + 4m_i(m_i+1) \cdot (f-y)^{(m_i)} \sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{x-a_j} \end{aligned}$$

$$2. \quad B \equiv - \frac{d^\alpha}{dx^\alpha} (d(x)(f-y)) = - \sum_{r=0}^{\alpha} \binom{\alpha}{r} (d(x))^{(r)} (f-y)^{(\alpha-r)}$$

B is now evaluated at a_i . For $0 \leq \alpha < m_i$, $B = 0$

$$\text{for } \alpha = m_i,$$

$$B = -d(x)(f-y)^{(m_i)} = -m_i(m_i+1)(f-y)^{(m_i)}$$

for $\alpha = m_i + 1$,

$$\begin{aligned} B &= -(d(x)(f-y))^{(m_i+1)} + (m_i+1) d'(x)(f-y)^{(m_i)} \\ &= -m_i(m_i+1)(f-y)^{(m_i+1)} - (m_i+1) \cdot 2 \cdot m_i \cdot (f-y)^{(m_i)} \sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{(x-a_j)} \end{aligned}$$

$$3. \quad C = -2 \frac{d^\alpha}{dx^\alpha} ((x-a_i) h(x) (f'-y')) = -2 \sum_{r=0}^{\alpha} \binom{\alpha}{r} [(x-a_i) h(x)]^{(r)} (f-y)^{(\alpha+1-r)}$$

Again, for $0 < \alpha < m_i$, $C = 0$ since whenever $r = 0$, $x-a_i = 0$.

for $\alpha = m_i$

$$C = -2m_i \cdot h(x) \cdot (f-y)^{(m_i)} = -2m_i^2 (f-y)^{(m_i)}$$

for $\alpha = m_i + 1$

$$\begin{aligned} C &= -2(m_i+1) \cdot h(x) \cdot (f-y)^{(m_i+1)} \\ &\quad - 2 \cdot \frac{(m_i+1)(m_i)}{2} \cdot 2 \cdot h'(x) \cdot (f-y)^{(m_i)} \\ &= -2(m_i^2+m_i)(f-y)^{(m_i+1)} - 2(m_i^2+m_i)(f-y)^{(m_i)} \sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{x-a_j} \end{aligned}$$

and

$$4. \quad D = \frac{d^\alpha}{dx^\alpha} [(x-a_i)^2 (f''-y'')] = \sum_{r=0}^{\alpha} \binom{\alpha}{r} ((x-a_i)^2)^{(r)} (f-y)^{\alpha+2-r}$$

Since $D = 0$ when $r < 2$, $D = 0$ for $0 \leq \alpha < m_i$

for $\alpha = m_i$

$$D = m_i(m_i+1)(f-y)^{(m_i)}$$

for $\alpha = m_i + 1$

$$D = (m_i + 1) \cdot m_i \cdot (f-y)^{(m_i+1)}$$

Combining A, B, C, and D for $\alpha = m_i$, the results are

$$S = [2m_i^2 - m_i^2 + m_i - 2m_i^2 + m_i^2 - m_i] (f-y)^{(m_i)} = 0$$

for $\alpha = m_i + 1$

$$\begin{aligned} S &= [2m_i^2 - m_i^2 + m_i - 2m_i^2 - 2m_i + m_i^2 + m_2] (f-y)^{(m_i+1)} \\ &\quad + [4m_i^2 + 4m_i - 2m_i^2 - 2m_i - 2m_i^2 - 2m_i] \left(\sum_{\substack{j=1 \\ j \neq i}}^k \frac{m_j}{x-a_j} \right) \\ &= 0 \end{aligned}$$

Therefore, $G''(x)$ is continuous.

Once the functions G' , G'' , G''' , and G'''' have been shown to be continuous, it is possible to obtain a value for each of them. Since the computations are again very similar, it will be shown for one of the four.

Consider the case of $G''''(x)$. Define a function $H(z)$ by

$$\begin{aligned} H(z) &= (G(x) - G(z) + G'(x)(z-x) + \frac{1}{2} G''(x)(z-x)^2 \\ &\quad + \frac{1}{6} G'''(x)(z-x)^3 + \frac{1}{24} G''''(x)(z-x)^4) \cdot P(z) \end{aligned} \quad (26)$$

Wendroff [2] proves a generalized Rolle's theorem which states that if $f(x)$ has an n -th derivative in $[a, b]$, $n \geq 1$, and if $f(x)$ has a zero of at least m_i at a_i , where $a \leq a_1 \leq a_2 \leq \dots \leq a_k \leq b$, and if

$\sum_{i=1}^k m_i \geq n+1$, then there exists an $\eta \in [a, b]$ such that $f^{(n)}(\eta) = 0$.

In the case at hand, $H(z)$ has at least $n+5$ zeros including five at $z = x$, and n associated with $P(z)$. This would mean that there is an η such that $H^{(n+4)}(\eta) = 0$. But on taking the derivatives, after multiplying out the right side of (26) and noting that $P(z) G(z) = f(z) - y(z)$, the function becomes

$$H^{(n+4)}(\eta) = f^{(n+1)}(\eta) + \frac{(n+4)}{4!} G'''(x) = 0$$

Solving this expression for $G'''(x)$ gives

$$G'''(x) = \frac{4!}{(n+4)!} f^{(n+4)}(\eta)$$

The general form would then be

$$G^{(j)}(x) = \frac{j!}{(n+j)!} f^{(n+j)}(\eta)$$

In order to know the value of the error estimates given in (18), the values P, P', P'', P''', P'''' need also to be obtained. Since the polynomials in the present case are the Lagrangian, Hermite and trinite polynomials, $P(x)$ takes on the form $p(x), p^2(x), p^3(x)$ where $p(x) = (x-a_1)(x-a_2)\dots(x-a_n)$. The derivatives of these polynomials are given in Table 1.

	<u>Lagrangian</u>	<u>Hermite</u>
$P'(x)$	$p'(x)$	$2p(x)p'(x)$
$P''(x)$	$p''(x)$	$2[(p'(x))^2 + p(x)p''(x)]$
$P'''(x)$	not needed	$2[3p'(x)p''(x) + p(x)p'''(x)]$
$P''''(x)$	not needed	not needed

	<u>Trimite</u>
$P'(x)$	$3p^2(x)p'(x)$
$P''(x)$	$6p(x)(p'(x))^2 + 3p^2(x)p''(x)$
$P'''(x)$	$6(p'(x))^3 + 18p(x)p'(x)p''(x) + 3p^2(x)p'''(x)$
$P''''(x)$	$36(p'(x))^2p''(x) + 18p(x)(p''(x))^2 + 24p(x)p'(x)p'''(x) + 3(p(x))^2p''''(x)$

Table 1. Expressions for Error Terms

Considering that $p(x) = 0$ and $p''(x) = 0$, since the calculations are done at the center point of an odd number of interpolating points (in this case $n=3$), the substitution of the values of Table 1 into (18) yields

1. for Lagrangian polynomials

first derivative error estimate is

$$\frac{p_3'(x)}{3!} f^{(3)}(x) \quad (27a)$$

second derivative error estimate is

$$\frac{2p_3'(x)}{4!} f^{(4)}(x) \quad (27b)$$

2. for Hermite polynomials

second derivative error estimate is

$$\frac{2(p'_3(x))^2}{6!} f^{(6)}(n) \quad (28a)$$

third derivative error estimate is

$$\frac{6(p'_3(x))^2}{7!} f^{(7)}(n) \quad (28b)$$

3. for trimite polynomials

third derivative error estimate is

$$\frac{6(p'_3(x))^3}{9!} f^{(9)}(n) \quad (29a)$$

fourth derivative error estimate is

$$\frac{24(p'_3(x))^3}{10!} f^{(10)}(n) \quad (29b)$$

The values of these estimates are then put into equations like (16) in place of e_1 and e_2 to determine $d_x(x_i, t)$. In the case of the Hermite or trimite, $d_x(x_i, t)$ would be found by using equations (12) and (13) for the development of $d_x(x_i, t)$.

In the time direction, $d_t(x_i, t)$ can be found from the automatic program used to solve the system of equations generated by the discretization.

Once these error estimates have been determined, there are two ways that the overall error could be approached. First, the sum of the two could be used by asking that during any time step this sum would be less than the user's requested error. This would be saying to the user that the equation being solved is close to his. However, the actual error in the solution will depend on the nature of the equation. If, therefore, the

user desired a particular error in the solution, he would need to have some idea about the nature of the solution so that a value of the error request can be obtained. This procedure would take less computer time but would involve more thought for the user.

The second approach would be to attempt to obtain an estimate of the error in the solution. This could be done by numerically solving a system of asymptotic error equations along with the discrete system of differential equations. The system of equations being solved is of the form

$$\frac{du_i}{dt} = \hat{f}(u_i, A_{n,x}(u_i), A_{n,xx}(u_i), x_i, t_k)$$

The system differs from the desired system in that the errors d_t and d_x have been introduced. This would lead to a system

$$\frac{\partial u(x_i, t)}{\partial t} = \hat{f}(u(x_i, t), A_{n,x}(u(x_i, t)), A_{n,xx}(u(x_i, t)), x_i, t) + d_t + d_x$$

Subtracting and letting $e_i(t) = u_i(t) - u(x_i, t)$, the equations obtained are

$$\frac{de_i}{dt} = \hat{f}(u_i, A_{n,x}(u_i), A_{n,xx}(u_i), x_i, t_k) e_i - d_t - d_x \quad (30)$$

Since the solution to this equation need not be accurate, it may be possible to use a lower order method to solve this last system provided $f_{\underline{u}}(\underline{u}(t))$ is the same for both systems.

The advantages of the latter method led to solving the system (30) along with the original system.

2.3 Efficiency

Once methods are available for solving the problem and for estimating the errors, attention needs to be given to producing an efficient system. That is, the solution needs to be found in such a way that it has certain error properties and such that the work involved in obtaining that solution satisfies some sort of minimization of computer time. If greater accuracy is desired, it is expected that more computer time will be used. If there is a decrease in computer time, then provided the overhead is always kept at a minimum, a loss of accuracy is expected.

The goal would be to have the error in the result less than, but as close as possible to, the amount requested by the user in such a way that no more computer time is used than is absolutely necessary. The primary emphasis needs to be that of keeping any estimated error less than the user's request.

If for some reason the estimated error becomes too large, overhead will have to be expended to rectify the problem. If the error becomes much smaller than the user's request, it might be desirable to modify the method--provided the overhead is less than the expected savings in time.

There are three ways to satisfy these goals:

1. modify $d_t(x_i, t)$ by changing the error request given to the subprogram solving the system of equations (DIFSUB),
2. modify $d_x(x_i, t)$ by changing the spacing in the partition,
3. modify $d_x(x_i, t)$ by changing the order of the spatial analog.

When beginning the integration, it would be desirable to determine first (if possible) those items involving the greatest amount of overhead. After the integration has proceeded a number of steps, the item involving the least amount of overhead or the item least apt to adversely affect the solution should be changed first. Care must also be taken that changes do not occur too frequently since modifications will, in general, have a tendency to introduce additional errors.

Of the three items above, the first has the most desirable properties. The change of this parameter could cause the vectors in DIFSUB to be multiplied by some factor depending on the change required in step size. On the other hand, the last two involve a greater amount of overhead and also involve interpolation that could introduce larger errors than those involved with changing the time parameter.

Consider the problem of obtaining an initial partition or of changing to a new partition. The problem is to determine the order of the method and the points to include in the partition so that the integration may be accomplished most efficiently and yet have the desired accuracy. If it is assumed that given a partition, the majority of work is involved with the function evaluation or calculation of derivatives, and in the solution of a system of equations required by DIFSUB, it would be possible to develop a sort of work estimate based on the number of multiplications and divisions present.

In the case of the derivative calculation, the two things to consider are the calculation of the derivative and the calculation of its error estimate. In the first case, we are using the Lagrangian, Hermite, and trimite three-point formulas, so that the coefficients are known as soon as

the step size is given. In the case of the error estimate, the coefficients for its interpolating polynomial are calculated and then stored for a given partition. When this estimate is calculated, there will be four interpolating points for each derivative (with the exception of the second derivative using the Lagrangian method which requires five points). The following table lists the number of operations used in calculating the derivatives for each method. The derivatives requiring calculation are the first and second for the Lagrangian, second and third for Hermite, and third and fourth for the trimite.

	<u>Multiplications/Divisions In Derivative</u>		<u>Multiplications/Divisions In Error Estimate</u>	
	lower	higher	lower	higher
Lagrangian	1	2	4	5
Hermite	4	5	8	8
trimite	7	8	12	12

Table 2. Number of Operations in Equation Evaluation

To aid in determining which method would be most efficient, the user is to indicate the derivatives required for each of the methods used.

In solving the system of equations, the procedure is divided into two steps:

1. decomposing the matrix into upper and lower triangular matrices,
2. using the decomposed matrix to solve the system of equations.

In each of these steps, the number of nonzero multiplicative operations required for each equation is proportional to one-half the number of off-diagonals. In the case of decomposing the matrix, this proportionality

constant is two. In the case of the solving step, the constant is three. The number of off-diagonals is two, five, and eight for the Lagrangian, Hermite, and trimite, respectively. The average number of operations per equation for each method with each step is given in Table 3.

	<u>Decomposition</u>	<u>Solving</u>
Lagrangian	2	3
Hermite	5	7.5
trimite	8	12

Table 3. Number of Operations in Decomposing and Solving

The problem in trying to bring the three sets of numbers together is that the relative frequency of the number of function (N_f), decomposition (N_d), and solving (N_s) calls depends on the nature of the problem. The number of function evaluations can be given directly as a function of N_d and N_s by

$$N_f = n_m * N_d + \frac{1}{2} N_s$$

where n_m is the number of equations in the system. The reason for the factor of $\frac{1}{2}$ is that for each call to obtain the function evaluation, the solving routine is called for use by the system of differential equations and for use by the system of asymptotic error equations. The first term arises from the fact that DIFSUB calculates a Jacobian by a numerical differencing method that calls for the function evaluation for each equation in the system. The ratio (R) of N_d to N_s is variable and depends on the solution to the system of equations. In systems where the Jacobian varies slowly, the ratio ranged from about .033 to .045. Since those equations were very well behaved and since an increased ratio indicates an increased

amount of work, a value of .05 will be used initially. The ratio will then be modified as the integration progresses. At the start of the integration, the value N_d could be written as

$$N_d = .05 N_s$$

An estimate of the work required for each method can now be obtained as a function of the number of calls to the solving routine. Such an estimate is given by

$$W_m = n_m \left\{ \frac{1}{2M} W_{f1} + R(W_{f2} \cdot \frac{n_m}{M} + W_d) + W_s \right\} \cdot N_s$$

where n_m is the number of equations, W_d and W_s are the number of operations in the decomposition and solving subroutines. M is the number of values stored at each point, and W_{f1} , W_{f2} are the operations required in a function call for a step in the solution, and a function call in determining the Jacobian. The latter distinction is brought about because the process of calculating the Jacobian does not require any error estimate calculations.

In order to decide which method should be used, the values of W_m/N_s for each of the methods are compared. Although one is interested in using the method with the smallest value of W_m/N_s , there is an additional complication in trying to decide which method to use. This involves the fact that if an attempt were made to change the partition during the integration to a higher ordered method, the interpolation polynomials would be such that error estimates would be too large to work with. As a result, it might be appropriate to bias the selection slightly in favor of the higher ordered methods during the initial partitions selection. After integration has begun, if a change is to be made in the partition, the simpler lower ordered methods would be preferred.

Once the partition has been selected, it would then be possible to adjust d_t to some extent so as to keep the error close to the user's request. If DIFSUB is asked to produce a large d_t then it will increase the step size or lower the order of integration. If asked to produce a smaller error, it will decrease the step size.

The modification of d_t and d_x should not occur too frequently since the introduced errors could become large enough that a satisfactory solution is impossible.

2.4 Initial Partition

In order to start the integration, it is necessary to have an initial partition. Since any modification in the partition will introduce additional errors, it is important to find a partition that will be the basis of an efficiently obtained solution that will satisfy the user's request. One way to develop such a partition would be to control the amount of error present in the derivative. This would lead to trying to obtain a spatial error parameter that would be used to calculate this initial partition.

There are three basic approaches that could be followed. One, the user could be asked to provide this parameter. Two, a set function of the user's overall error request could be used to develop such a parameter. Three, an arbitrary set of partitions using equal spacings could be set up and then tested to see what error results in the asymptotic error formula.

The first method of the three has the least desirable properties from the standpoint of the user. It requires that he make some sort of

judgment about the equation. If his judgment gives poor results, then the parameter would have to be modified. Since the program would eventually have to modify this parameter, it would be appropriate to generate the parameter initially.

The second method has the advantage that the function could be an easily calculated value. For example, one could have this parameter be one-tenth the value of the user's request. If this generates a partition that creates valid errors, then no unnecessary work has been done. If the errors generated are too large, then the partition would have to be recalculated based on some formula of parameter modification.

The third method would start with a partition using a step size of $\frac{a-b}{2^m}$, where a and b are the end points. Using this partition, the integration would be performed over one step. The error in the solution would then be compared with the errors in the derivatives. The process would be continued until the errors in the one step of integration correspond with the user's request. The maximum error estimate in the derivatives would then be used to generate the working partition. The disadvantage with this approach is that overhead will always exist. If the first partition yields a value for the spatial error parameter, the working partition would still need to be generated. The exception to this would be the case where the working partition contains equal spacings.

Since the second method is simpler to work with and since it can, if required, perform somewhat like the third method, it was decided that the second approach would be used.

Once an error parameter has been determined, the working partition needs to be generated. Two approaches were considered. One, it would be possible to start at one of the end points and then choose each successive point so that the error condition is met. Two, it would be possible to start with three intervals and then subdivide each interval in which the error condition is not met.

In the first method, one would start by adding two points to the partition, a center point and a second end point of the first subinterval. These two points would be chosen so that the error condition would be met. The next step would be to use the second point above as the center of a new subinterval, and if necessary, add points on either side of this point so that the symmetric three-point formula would again satisfy the error condition. The process is continued until no new subintervals need to be added to completely cover the entire interval $[a,b]$.

Some of the disadvantages of this procedure are

1. each time a new center point is added, it may be necessary to backtrack in the interval to handle newly added points,
2. even if all step sizes used in the subinterval are powers of two, there may develop problems terminating at the second endpoint,
3. the possible backtracking could create programming complications.

In the second method, start with the three points $P_1 = a + \frac{b-a}{2}$, $P_2 = a + \frac{b-a}{4}$, $P_3 = a + 3 \cdot \frac{b-a}{4}$. (see Figure 1)

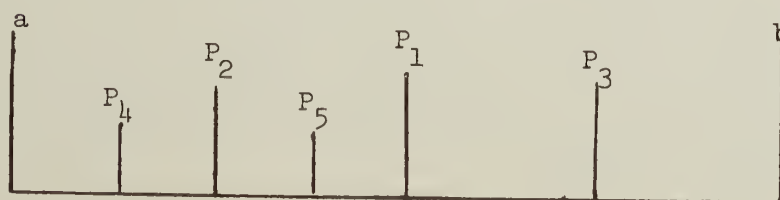


Figure 1

The subinterval $[a, P_1]$ is next checked to see if the error is reasonable. If not, points are added on either side of P_2 with step size $\frac{b-a}{8}$. If the error was acceptable, then this fact is recorded so that on future passes through the interval, this subinterval will be bypassed. The next step would be to check the subinterval $[P_2, P_3]$ to see if the analog for P_1 satisfies the error condition. At the same time, the newly added points P_4 and P_5 are checked to make sure that they also satisfy the error condition. This process is repeated until all subintervals satisfy the error condition.

In order to find out if a given derivative satisfies the error requirements, it is necessary to generate some sort of error estimate. An attempt is made to duplicate the type of error estimate that would be made in the actual integration.

In an attempt to guard against unusual initial conditions, the testing at each point is done twice--once with an $h = \frac{b-a}{2^n}$ and once with $h = \alpha \frac{b-a}{2^n}$ --where $.9 < \alpha < 1$.

2.5. Changes in Integration

Once the integration parameters have been set, nothing needs to be changed until one of two conditions occur. If the estimated error ever becomes larger than the desired value, then some sort of corrective action will be required immediately. If the estimated error becomes much smaller than the requested value, it may indicate that a change is desirable.

Consider the case where the estimated error is larger than desired. There are two possible actions--either the error parameter to DIFSUB or partition could be changed.

The first action to be taken would be to attempt to change d_t . This would be done provided the difference between the estimated error and the error request is smaller than the error reduction possible from DIFSUB by changing d_t . If this condition cannot be met, or if several changes in this parameter have already occurred, then it will be necessary to change the partition.

Although it would be possible to add selected points, it was felt that since the changing of the partition is being done as a last resort, the desirable action would be to change the entire partition to one with smaller step sizes. There are two other advantages in this course of action. First, the program used to change the partition could be basically the same as that needed to modify it. Second, since the entire partition now is satisfying a smaller value for d_x , there is the possibility that the partition would not need to be changed as frequently.

If the conditions above occur during the first several steps, then it is assumed that a finer partition should have been used initially. Rather than create a new partition, it was felt that a better solution would be obtained if the process were restarted with a smaller value in the error parameter. If this process of restarting must be repeated a few times, then it is assumed that the problem cannot be solved with this method.

If the estimated error is much smaller than that required, then a similar sequence of steps would occur as those just discussed. The first step would be to see if a change in the time error parameter is appropriate. The maximum size allowed for the time error parameter would be approximately nine-tenths of the user's request. If the error estimate continues to remain small, then consideration will be given to changing the partition.

Since going to new partitions can introduce errors, there is an attempt to delay any changes to coarser partitions unless there is ample indication that such a partition will not have to be changed back.

Just as in going to finer partitions, it would be possible to set up the program to delete selected points where the error estimates indicate it could be done. However, since precautions are taken in not changing too soon, it is felt that when a change is made to a coarser partition, the entire partition should be changed. Also, it would be appropriate to see if a lower ordered method could be used. The ideal situation in trying to go to a lower ordered method would be to be able to keep the same partition but just reduce the number of items saved at each point. Normally, when changing to a lower ordered method, one would expect to have to interpolate some in order to make up for the loss of values at the partition points.

2.6 The User's Equations

The communication between the user and this automatic system of subprograms used to solve the user's equation (called the system) should be kept as simple as possible. Besides a program to drive the system, the user would be expected to provide subprograms that give the differential equation, initial values, and boundary values.

In the case of the differential equation, the user would be expected to provide an equation of the form

$$\frac{\partial u}{\partial t} = f(u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, x, t).$$

The system would provide the values for the arguments of the function f , and his program would return the time derivative.

If the higher ordered methods are also used, then two other equations would also be included. These equations would have the forms

$$\frac{\partial v}{\partial t} = g(u, v, \frac{\partial v}{\partial x}, \frac{\partial^2 v}{\partial x^2}, x, t)$$

and

$$\frac{\partial w}{\partial t} = h(u, v, w, \frac{\partial w}{\partial x}, \frac{\partial^2 w}{\partial x^2}, x, t)$$

where

$$g = \frac{\partial}{\partial x} f, h = \frac{\partial^2}{\partial x^2} f$$

and

$$v = \frac{\partial u}{\partial x}, w = \frac{\partial^2 u}{\partial x^2}$$

Again, the system would provide the values for the variables of each function, and the user would return the time derivatives.

A similar approach is used for the initial values. Considering that all ordered methods might be used, initial values would be required for u , v , and w . If $u(x,0) = r(x)$, then $v(x,0) = s(x) \equiv \frac{dr(x)}{dx}$ and $w(x,0) = t(x) \equiv \frac{d^2r(x)}{dx^2} = \frac{ds(x)}{dx}$. If the derivatives of one of these do not exist, then either an approximation with continuous derivatives will need to be developed or the method will have to be restricted to those lower ordered methods for which the initial values exist.

The problem of the boundary conditions is more difficult to handle. The general form of the boundary condition is

$$u(0,t) + u_x(0,t) - c(t) = 0 \quad \text{at } x = 0$$

and

$$u(1,t) + u_x(1,t) - d(t) = 0 \quad \text{at } x = 1$$

The user supplies these in terms of algebraic equations of the form

$$r = u_i + A(u_i, x_i, t) - d(t)$$

where $A(u_i, x_i, t)$ is some representation of u_x and r will be the error in satisfying the boundary condition.

If the Lagrangian method is used, the function A will have to be determined by some form of interpolation. In the case of the two higher ordered methods, $u_x \equiv v_i$ so that $A(u_i, x_i, t) = v_i$ could be used.

Since the higher ordered methods require additional values at the boundaries, it will be necessary for the user to provide additional equations involving these variables. It would be ideal if these boundary value equations could be obtained from the boundary condition of the original equation. The simple example of

$$u_t = u_{xx}, u(x,0) = \sin(\pi x), u(0,t) = u(1,t) = 0$$

shows that this will not work. The solution of this equation is given by $u(x,t) = e^{-\pi^2 t} \sin(x)$. From the definition of $v(x,t)$, the solution is found to be

$$v(x,t) = \pi e^{-\pi^2 t} \cos(\pi x)$$

Thus,

$$v(0,t) = -v(1,t) = \pi e^{-\pi^2 t}$$

which cannot be obtained from

$$u(0,t) = u(1,t) = 0$$

One possibility for handling the problem would be to use an equation of the form

$$r = v_i - B(u_i, v_i, x_i, t)$$

where B is an approximation of v_i which is developed from interior points.

3. PROGRAMS

The purpose of this section is to provide a description of the programs used in developing the automatic solution. There are eleven such programs. Eight of these programs (PARDIF, XSTP, DIFFUN, DERIV, DIVERR, DECØMP, SØLVE, DIFSUB) are set programs that make up the method. The remaining three (BDRY, INIT, UFUN) are user-supplied and provide information about the user's equation, initial values, and boundary conditions. The user also needs a driving program which would handle the printing.

In order to use the system, the driving program calls PARDIF for each step in the time direction. As a result, the user can determine when and what information he desires to print out, and when to stop the integration. These programs are in Appendix A.

3.1 PARDIF

PARDIF is the main program of the method. It has the basic purpose of integrating one step for each call made to it. In performing this step, it must cause an initial partition to be generated, create functional values for this partition, integrate the step, decide whether the step was acceptable, and decide whether modifications should be made.

The creation of a partition is done in two steps. First, it calls XSTP which provides as output a pointer to indicate the proper order of points, and a vector which contains partition values as well as an indication of the other points which are used in analog. The second step consists of placing values in the proper order in the array x and placing two values in a second array (INT). The one value indicates the relative

position of one of the other points in the analog. This will provide sufficient information since the partition is generated in such a way that two of the three points of the analog are adjacent. The pointer, therefore, is used to indicate the third point. The other value in INT is the starting position for the interpolating polynomial of the error estimate.

Once the partition is well-defined, the value of the function at each of these points needs to be found. If this is the initial partition, then the values are obtained from an initial value subprogram (INIT). If the partition is a modification of an existing one, the program tries to find the values in the old partition. If this attempt is unsuccessful, then the values are found by interpolation by calling a subprogram (DERIV). Since there are several different vector components that would use the same interpolation, DERIV is asked to return the coefficients of this polynomial. The interpolation of all components is then accomplished in PARDIF.

Every time a new partition is generated, the program calculates--by calling DERIV--the coefficients used to calculate the error estimates. These coefficients are then stored in an array (DIVER).

If all the values associated with the partition have been defined, or if this step does not modify any of the parameters, the program is ready to integrate and check the results. The integration is done by calling DIFSUB.

On returning from DIFSUB, the program first checks the return code to find out if the step was completed satisfactorily. If not, the program first tries to decrease d_t by decreasing the time based error parameter (TPS). If this procedure is unsuccessful, an attempt is made to change the partition.

Whenever the return code from DIFSUB indicates that the integration was successful, PARDIF determines whether any additional action is to be taken. To accomplish this, the program determines the maximum error (RMAX) introduced in the step by looking at the results from the asymptotic error equation.

First, consider the case where RMAX is larger than the user's request (EPS). First of all, the number of steps is checked to see if it is within ten steps of $t = 0$. If so, the time based error parameter (TPS), and the space based error parameter (SPS) are reduced by a factor of ten, and the integration is restarted from $t = 0$. This process is repeated three times before taking a terminal exit.

When RMAX is larger than EPS, an attempt is made to decrease TPS until it has a value less than 10^{-5} . Once TPS has reached this value, the space parameter (SPS) is decreased by a factor of .5 in an attempt to change the partition. The desire is to add enough points to cause the error estimate to be less than EPS.

In the event that RMAX is less than EPS, consideration can be given to modifying parameters so as to decrease the work. The program DIFSUB will automatically change the order of the method it uses and the step size associated with that method to keep the error close to parameter TPS.

No changes will be attempted in TPS or SPS unless two conditions are satisfied. One, the last modification has not occurred in the previous ten steps. Two, the error introduced in the step must be less than some α times EPS.

If the two conditions have been met, TPS is increased by the minimum of

$$e^{\frac{\log TPS + \log EPS}{2}} \text{ and } TPS + \frac{1}{2} (EPS - RMAX).$$

Although many other methods of changing are possible, this choice appears to change TPS rapidly without causing the errors to become too large.

After three such changes are made in TPS, consideration is given to changing the partition. If SPS is changed (and, therefore, the partition), an additional restriction is imposed. This restriction is that RMAX be less than $\alpha^2 \cdot EPS$. If this condition is met, then SPS is given the value of the minimum of $2 \cdot SPS$, $SPS + .5(EPS - RMAX)$ and $.4 \cdot EPS$. If a change does take place in SPS, then TPS will be given a new value by

$$TPS = .5 \cdot (EPS - SPS).$$

Again, the changes represent one choice of many possibilities. This time it is desirable to increase the parameter rather slowly so that the partition does not change too much.

Consideration will also be given to changing the partition without changing SPS provided the following set of conditions is satisfied,

1. at least 30 steps have occurred since the previous modification,
2. RMAX has remained less than EPS,
3. each component of the spatial truncation error is less than $\alpha^2 \cdot EPS$.

3.2 XSTP

XSTP has the function of generating the positional values of the partition under two different situations. The first occurs when the initial partition is generated and the second whenever an existing partition is modified.

In both cases, the program begins with partition points at .5, .25, .75, and a step size $h = \frac{1}{4}$. On each succeeding pass through the interval, h is divided by 2. Until $h \leq HMAX$ (the maximum step size the user desires), the program places points at $(2i-1) \cdot h$, $1 \leq i \leq \frac{1}{2h}$. After $h > HMAX$, the program places points only when the error estimate in the analog does not satisfy the error condition. These additional points are placed on either side of the test point. Because of the way the partition is generated, each point that is added will be halfway between two existing points.

If the analog at a particular point satisfies the error request, this fact is recorded in the vector NCK. Before trying to add points, the program first checks NCK to see if the testing can be ignored. When all values of NCK are set to +1, then the procedure is terminated.

The differences between the two uses of XSTP take place in obtaining the functional values and the error estimates. In obtaining the functional values for an initial partition, the program calls the user's initial value program (INIT). The case of modifying an existing partition is accomplished by the interpolation of values of the previous partition.

For estimating the error of an initial partition, the program adds two points whose positions relative to the points of the analog, are fixed. This allows the coefficients to be calculated for all cases. When the partition is changed, the functional values are not as easily determined, so the error estimate is obtained by using values determined for the partition and calling the subprogram DERIV to interpolate.

Once the partition for a particular order has been developed, the work estimate is obtained using the ideas of section 2.3. The entire process is then repeated for the next ordered method.

When the partitions for all possible orders have been developed, the work estimates are compared to determine which method should be used. This is done by asking that between two adjacent ordered methods, the smaller of αw_{m_1} , w_{m_2} (where m_1 is the lower order) indicates the order preferred. The α is used to set the biases desired in the selection of a method.

Once the method has been chosen, the program then checks each analog to make sure that the smallest available step size is being used. It also prepares output vectors indicating the natural order of points and the points that will be used in each analog.

3.3 DERIV

DERIV is a double purpose subprogram involved with interpolation. The one task is to calculate the coefficients for the error estimation polynomial of a given partition. The other task is to provide interpolation polynomials for calculating new values of a partition.

The error estimates needed are given by (27), (28), and (29). Each of these involves the factor $P'_n(x)$, which can be determined as a function of the spacing h . Since $n = 3$, $P_n(x) = (x-a_1)(x-a_2)(x-a_3)$, and $x = a_2$, the value of $P'_n(x)$ is given by

$$\begin{aligned} P'_n(x) &= (x-a_1)(x-a_2) + (x-a_1)(x-a_3) + (x-a_2)(x-a_3) \\ &= (x-a_1)(x-a_3) \\ &= -h^2 \end{aligned}$$

Thus, the error terms become

$$-\frac{h^2}{6} f^{(3)}(\eta) \text{ and } -\frac{h^2}{12} f^{(4)}(\eta) \text{ for the Lagrangian}$$

$$\frac{h^4}{360} f^{(6)}(\eta) \text{ and } \frac{h^4}{840} f^{(7)}(\eta) \text{ for the Hermite}$$

and

$$\frac{h^6}{60480} f^{(9)}(\eta) \text{ and } \frac{h^6}{151200} f^{(10)}(\eta) \text{ for the trimite}$$

The problem remaining in determining the estimate is the evaluation of $f^{(j)}(\eta)$. In the case of the Lagrangian polynomials

$$f^{(j)}(x) = \sum_{i=1}^n \ell_i^{(j)}(x) f(x_i) \quad (31)$$

In order to obtain an estimate of $f^{(3)}(x)$, and $f^{(4)}(x)$, the $\ell_i(x)$'s were chosen so that the proper derivatives existed. In the case of the third and fourth derivatives, 4 and 5 points were required, respectively. Using these polynomials gives

$$f^{(3)}(x) = \sum_{i=1}^4 \frac{3!}{\prod_{\substack{j \neq i \\ j=1}} (x-a_j)} f(a_i)$$

and

$$f^{(4)}(x) = \sum_{i=1}^5 \frac{3!}{\prod_{\substack{j \neq i \\ j=1}} (x-a_j)} f(a_i)$$

The program then generates the coefficients of $f(a_i)$ for use by other programs.

In calls from XSTP, the actual value is desired, so the coefficients are multiplied by $f(a_i)$ and the value returned.

In the case of the Hermite polynomials, the problem is to obtain estimates of $f^{(6)}(x)$ and $f^{(7)}(x)$ when given that

$$f(x) \approx \sum_{i=1}^n h_i(x) f(a_i) + \bar{h}_i(x) f'(a_i) \quad (32)$$

For this the derivatives of $h_i(x)$ and $\bar{h}_i(x)$ are desired. The results are given by (assuming four points)

$$h_i^{(6)}(x) = 20(1 - 2\ell_i'(a_i)(x - a_i))(\ell_i'''(x))^2 + 120(-2\ell_i'(a_i))(\ell_i''(x) \ell_i'''(x))$$

$$h_i^{(7)}(x) = 140(-2\ell_i'(a_i))(\ell_i'''(x))^2$$

and

$$\bar{h}_i^{(6)}(x) = 20(x - a_i)(\ell_i'''(x))^2 + 120\ell_i''(x) \ell_i'''(x)$$

$$\bar{h}_i^{(7)}(x) = 140(\ell_i'''(x))^2$$

In the case of the trimite polynomials, the desired estimates are $f^{(9)}(x)$ and $f^{(10)}(x)$. The polynomial $f(x)$ is given by

$$f(x) = \sum_{i=1}^n (t_i(x) f(a_i) + \bar{t}_i(x) f'(a_i) + \bar{\bar{t}}_i(x) f''(a_i)) \quad (33)$$

so that the ninth and tenth derivatives are desired for t_i , $\bar{t}_i$, and $\bar{\bar{t}}_i$. Each of these has the form $V = A(\ell(x))^3$ as developed in section 2.2 where V is one of t_i , $\bar{t}_i$, or $\bar{\bar{t}}_i$. $V^{(9)}(x)$ and $V^{(10)}(x)$ are then given by

$$\begin{aligned} V^{(9)}(x) &= (1680(\ell_i'''(x))^3) \cdot A + 15120 \ell_i''(x)(\ell_i'''(x))^2 \cdot A' \\ &\quad + 36(630(\ell_i''(x))^2 \ell_i'''(x) + 420\ell_i'(x)(\ell_i'''(x))^2) \cdot A'' \\ V^{(10)}(x) &= 16800(\ell_i'''(x))^3 \cdot A' + 75600\ell_i''(x)(\ell_i'''(x))^2 \cdot A'' \end{aligned}$$

It will be noted that in all of the cases above the estimates turned out to be functions of the Lagrangian polynomials. Therefore, the program

generates $\ell_i(x)$, $\ell_i'(x)$, $\ell_i'(a_i)$, $\ell_i''(x)$, $\ell_i''(a_i)$, $\ell_i'''(x)$ as needed and then combines them to form the desired estimate.

The second task of this subprogram is to interpolate to find values at new partition points. Since these interpolating polynomials are also functions of the Lagrangian polynomials, much of the code used above can also be used here.

Since new partitions will only involve the same or lower ordered methods, the Lagrangian method only requires (31) with $j = 0$. For the Hermite polynomials, the first derivative and the functional values are needed. The functional values are obtained from (32). Since h_i and $\bar{h}_i$ can be written in the form $V = A\ell^2(x)$ where A is $[1 - 2\ell_i'(a_i)(x - a_i)]$ and $(x - a_i)$, respectively, the first derivatives can be written using

$$V'(x) = A'\ell^2(x) + 2A\ell(x)\ell'(x)$$

Finally, the trimite method will require the first and second derivatives as well as the functional values. The functional values are obtained using (33). Using the form $V = A^3(x)$ for the t 's (see page 15 for values), the first and second derivatives can be obtained by

$$V'(x) = A'\ell^3(x) + 3A\ell^2(x)\ell'(x)$$

and

$$V''(x) = A''\ell^3(x) + 6A'\ell^2(x)\ell'(x) + 3A[2\ell(x)(\ell'(x))^2 + \ell^2(x)\ell''(x)]$$

Since calls to DERIV for purposes of interpolating new partition values may require the coefficients in some cases and values in others, both of these options are made available.

3.4 DIFSUB

This program is a modification of a program by Gear [3]. The modifications basically deal with four areas:

1. the presence of the asymptotic error equation,
2. the use of DECØMP and SØLVE,
3. the use of algebraic equations for the boundary conditions,
4. the presence of externally changed parameters.

Probably the largest number of modifications deals with the area of the asymptotic error equations. The main item here is that every time something is done with the solution vector (Y), the same thing is also done with the asymptotic error system.

Since the solution to the asymptotic error system (HMY) does not need to be as accurate as Y, coefficients were developed by Gear [4] so that the first two coefficients would be the same in all methods, and thus, allow the asymptotic error system to use a lower ordered method.

This can be accomplished since the solution involves the Newton iteration

$$z_{n,(m+1)} = z_{n,(m)} + \ell \left[-\frac{\partial F}{\partial a_i} \ell \right]^{-1} F(z_{n,(m)}) \quad (34)$$

where $F(z_{n,(m)}) = hf(\underline{u}) - hu'$ in this formula,

$$W = \left[-\frac{\partial F}{\partial a} \ell \right]^{-1} = [\ell_1 - h\ell_0 \frac{\partial f}{\partial u}]^{-1}$$

where ℓ_0 and ℓ_1 are the first two coefficients of the method. Since W is the same for all ordered methods, the two systems can be solved using different orders.

Another slight modification required by the asymptotic error system is that in order to start integrating, initial time derivatives are required. This means that the evaluation of the Jacobian must be present earlier in the program.

The need for the second modification is discussed in Gear [3]. He noted that the break point for using DECØMP and SØLVE occurred when about five or more equations are in the system. Since the number of equations solved here is in general eight or more, the need is apparent. As an added reason, the presence of the asymptotic error equations doubles the number of calls to SØLVE whereas the number of calls to DECØMP will remain about the same.

Two changes are required by the presence of algebraic equations for the boundary conditions. First, in equation (31) the function F is replaced by $G(u_i)$ which is the residual of the algebraic equation. Second, the calculation of those rows of W involved with algebraic equations will need to be modified. In this case, the component is given by

$$\left[-\frac{\partial G}{\partial a_i} \underline{l} \right]^{-1} = \left[-\underline{l}_0 \frac{\partial g}{\partial a_0} \right]^{-1}$$

where g is the algebraic equation. The absence of $\underline{l}_1$ is due to the fact that derivatives of the variables are not present in G (see Gear [5]).

In the fourth area, there are two conditions that take place:

1. change in the error parameter
2. change in the partition.

In the first of these, the only requirement is that various error condition values be recalculated. As a result, a jump is made to that portion of the program.

The second is more involved. If the partition has been changed, then it will be necessary to change those variables dealing with the number of equations in the system, and to re-evaluate the Jacobian.

3.5 DECØMP and SØLVE

The purpose of these programs is to solve $Y = Ax$ for x where A is a banded matrix. The procedure is carried out in two steps. (The programs are modification of ones given by Forsythe and Moler [6].) The first step is to decompose the matrix in such a way that the lower triangular portion will represent the operations done to the vector Y in the first half of a Gaussian elimination procedure, and the upper triangular portion will represent the coefficients of the variables used in the back substitution process. This step is done by DECØMP and is called each time the matrix A is recalculated.

The second step consists of two parts. The first takes the lower triangular portion and carries out the operations of the first half of the Gaussian elimination on the vector Y . The second part performs the back substitution.

In these subprograms, the matrix A is stored in a double precision array (UL) diagonal by diagonal, with the main diagonal stored in $UL(\frac{K7+1}{2}, I)$ where $K7$ is the number of diagonals and is assumed to be odd.

3.6 DIFFUN

DIFFUN calculates the time derivatives of the system based on the differential equation. It does this by calculating the various space

derivatives required and then supplies these values to the differential equation subprogram (UFUN). At the same time, it also calculates the error estimate of the space derivatives by using the stored coefficients of the interpolating polynomials for such estimates. These results are then added to the derivatives and given again to UFUN. The difference between the results of the two calls to UFUN then provides an estimate of $d_x(x_i, t)$.

A second use of DIFFUN is to help determine the Jacobian used in DIFSUB in the predictor-corrector method and in the differential equation for the asymptotic error. In this environment, the error estimates are not used so a variable is set which causes this calculation to be bypassed.

3.7 DIVERR

The asymptotic error equation was given by (section 2.2)

$$e'(t) = f_y(y(t)) \cdot e(t) - d_t - d_x$$

This program calculates the time derivative $e'(t)$ for each variable using $f_y(y(t))$ which is stored in the array PEQ, d_t which is given by

$\frac{ERROR(IA)}{PERTST}$, and d_x which is stored in ERR(IA).

4. RESULTS

The system was used successfully to examine three separate examples. In each case, the problem was solved using the system and an alternative numerical technique. The three problems were:

$$\begin{aligned} 1. \quad u_t &= u_{xx}, \quad u(0,t) = u(1,t) = 0, \\ u(x,0) &= \sin(\pi x) \end{aligned} \quad (35)$$

$$\begin{aligned} 2. \quad u_t &= u_{xx} + u_x + u, \quad u(0,t) = u(1,t) = 0, \\ u(x,0) &= e^{-x} \sin(\pi x) \end{aligned} \quad (36)$$

$$\begin{aligned} 3. \quad u_t &= u_{xx}, \quad u(0,t) - u_x(0,t) = 0 \\ u(1,t) + u_x(1,t) &= 0, \quad u(x,0) = \sin^2(\pi x) \end{aligned} \quad (37)$$

Example 1

In this example, the solution is quite simple to calculate and is given by

$$u(x,t) = e^{-\pi^2 t} \sin(\pi x) \quad (38)$$

The independent solution uses a method developed by Crank and Nicholson [7]. The programming of the method is based on information in Richtmyer and Morton [8]. The program is listed in Appendix B.

In solving the problem with the latter method, an answer was obtained using a step size of $\Delta x = \frac{1}{8}$ and $\Delta t = \Delta x$. The process was then repeated with step sizes of $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$, ... until the error was less than .001. Results are in Table 4.

t	$x = \frac{1}{8}$		$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$		number of steps
	solution	error	solution	error	solution	error	solution	error	
.125	.111	-.25E-3	.205	-.45E-3	.268	-.59E-3	.291	-.64E-3	8
.25	.032	-.14E-3	.060	-.26E-3	.078	-.34E-3	.084	-.37E-3	16
.5	.27E-2	-.24E-4	.50E-2	-.45E-4	.66E-2	-.58E-4	.71E-2	-.63E-4	32
1.	.19E-4	-.34E-6	.36E-4	-.63E-6	.47E-4	-.84E-6	.51E-4	-.90E-6	64
2.	1.E-9	.3E-10	.2E-8	-.6E-10	.2E-8	-.8E-10	.25E-8	-.9E-10	128
4.	2.E-17	-.3E-18	.5E-17	.5E-18	.7E-17	.3E-18	.7E-17	-.2E-18	256
8.	-.2E-20	-.2E-20	-.2E-20	-.2E-20	.1E-19	.1E-19	.3E-20	.3E-20	512

Table 4. Solution to Example 1, Maximum Error .001, Using Crank-Nicholson Method

In using the system, answers were obtained with EPS equal to .1, .01, and .001. The errors in the first two cases are approximately the same since the same initial partition was used in both. The results are given in Tables 5, 6, and 7. In these tables, the column headed "Number of Equations Evaluated" represents the number of calls to UFUN times the number of equations per partition point.

The solution for $EPS = .001$ when using the system took approximately 26 seconds. If it had been known that $\Delta x = \frac{1}{64}$ would have produced the desired solution, about seven seconds would have been required. It would be normal in developing the latter solution to compare the results obtained with two different step sizes to estimate the error. If this were done using $\Delta x = \frac{1}{32}$ and $\frac{1}{64}$, the difference would have been about .002. Since the truncation error is $O[(\Delta t)^2] + O[(\Delta x)^2]$ [8], it would be expected that the error in the solution using $\Delta x = \frac{1}{64}$ would be near .0005 which would be within the .001 desired. This process would have required about eight seconds.

If the difference between the two solutions themselves must be less than .001, then step sizes of $\frac{1}{64}$ and $\frac{1}{128}$ would have been required. Using these two step sizes, the result would have required approximately 32 seconds.

The first method for obtaining the solution would require $\frac{1}{3}$ as much time as that obtained from the system. The second method would have required more time than the system to obtain the solution.

t	x = $\frac{1}{8}$				x = $\frac{1}{4}$				x = $\frac{3}{8}$				x = $\frac{1}{2}$				# of eqs. evaluated	number of steps	number of eqs./point
	solution	error	solution	error	solution	error	solution	error	solution	error	solution	error	solution	error	solution	error			
.121	.114	-.12E-2	.211	-.21E-2	.276	-.29E-2	.298	-.30E-2	1620						4			2	
.22	.042	-.15E-2	.077	-.27E-2	.101	-.36E-2	.109	-.39E-2	2070						6			2	
.48	.24E-2	-.96E-3	.45E-2	-.18E-2	.59E-2	-.23E-2	.64E-2	-.25E-2	2402						9			2	
1.09	-.21E-3	-.21E-3	-.38E-3	-.40E-3	-.50E-3	-.52E-3	-.54E-3	-.56E-3	2952						11			2	
1.56	.84E-4	.84E-4	.15E-3	.15E-3	.20E-3	.20E-3	.22E-3	.22E-3	2988						12			2	
4.45	-.2E-4	-.2E-4	-.4E-4	-.4E-4	-.55E-4	-.55E-4	-.6E-4	-.6E-4	3492						15			2	
8.45	-.4E-5	-.4E-5	-.8E-5	-.8E-5	-.1E-4	-.1E-4	-.1E-4	-.1E-4	3708						19			2	
16.45	.4E-8	.4E-8	-.2E-6	-.2E-6	-.4E-6	-.4E-6	-.5E-6	-.5E-6	4140						27			2	

Table 5. Solution for Example 1, EPS = .1, Using the System

t	$x = \frac{1}{8}$				$x = \frac{3}{8}$				$x = \frac{1}{2}$				# of eqs. evaluated	number of steps	number of eqs./point
	solution	error	solution	error	solution	error	solution	error	solution	error	solution	error			
.120	.116	-.42E-3	.217	-.76E-3	.281	-.98E-3	.304	-.11E-2					2142	8	2
.235	.037	-.44E-3	.069	-.81E-3	.090	-.10E-2	.097	-.11E-2					3042	12	2
.51	.24E-2	-.16E-3	.45E-2	-.30E-3	.58E-2	-.40E-3	.63E-2	-.43E-3					4014	18	2
.95	.12E-4	-.21E-4	.22E-4	-.40E-4	.29E-4	-.52E-4	.31E-4	-.56E-4					4464	22	2
2.01	-.17E-6	.17E-6	-.38E-6	-.38E-6	-.49E-6	-.49E-6	-.53E-6	-.53E-6					5346	25	2
4.03	.2E-7	+.2E-7	-.1E-7	-.1E-7	-.2E-7	-.2E-7	-.2E-7	-.2E-7					5858	29	2
8.23	.3E-7	.3E-7	.4E-10	+.4E-10	-.1E-9	-.1E-9	.9E-10	.9E-10					6156	37	2
15.7	.3E-7	.3E-7	.5E-10	.5E-10	-.8E-10	-.8E-10	.1E-9	.1E-9					7038	51	2

Table 6. Solution for Example 1, EPS = .01, Using the System

t	$x = \frac{1}{8}$		$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$	# of eqs. evaluated	number of steps	number of eqs./point
	solution	error	solution	error	solution	error	solution	error		
.128	.108	-.22E-4	.120	-.26E-4	.261	-.28E-4	.282	-.29E-4	5454	49
.245	.034	-.35E-4	.063	-.59E-4	.082	-.75E-4	.089	-.81E-4	5994	62
.49	.30E-2	-.28E-4	.56E-2	-.52E-4	.73E-2	-.67E-4	.78E-2	-.73E-4	7380	81
1.03	.11E-4	-.3E-5	.20E-4	-.58E-5	.26E-4	-.76E-5	.28E-4	-.83E-5	8802	89
1.98	.53E-9	-.63E-9	.99E-9	.12E-8	.13E-8	-.15E-8	.14E-8	-.16E-8	9434	101
3.75	.6E-11	.6E-11	-.6E-12	-.6E-12	-.2E-11	-.2E-11	-.2E-11	-.2E-11	10190	116
8.32	-.5E-11	-.5E-11	+.6E-12	.6E-12	.1E-11	.1E-11	.8E-12	.8E-12	10640	121
16.32	-.4E-11	-.4E-11	.8E-12	.8E-12	.6E-12	.6E-12	.3E-12	.3E-12	10928	129

Table 7. Solution for Example 1, EPS = .001, Using the System

Example 2

Although the equation (36) can be put into the form of equation (35) by the substitution $u = e^{-x}v$, it was felt an example involving other terms would be desirable. Because of the transformation and (38), the solution to (36) is seen to be

$$u(x,t) = e^{-x} e^{-\pi^2 t} \sin(\pi x)$$

The program used to obtain the independent solution is listed in Appendix C. The results of this program are given in Table 8. The results obtained from the system for EPS = .1, .01, and .001 are in Tables 9, 10, and 11, respectively.

For the independent solution, a step size $\Delta x = \frac{1}{256}$, $\Delta t = \Delta x$ was required before the solution was in error by no more than .001. However, the largest difference between solutions with $\Delta x = \frac{1}{64}$, $\Delta x = \frac{1}{128}$ was .0015 which would indicate that $\Delta x = \frac{1}{128}$ could be used. If $\Delta x = \frac{1}{64}$ and $\Delta x = \frac{1}{128}$ were used to determine the solutions, 31 seconds would be used. If $\Delta x = \frac{1}{128}$, and $\Delta x = \frac{1}{256}$ were used, then 124 seconds would be required. The solution using EPS = .001 with the system also requires about 31 seconds.

Example 3

The system can also be used to solve problems where the boundary conditions are of a more general form. In this example, the boundary conditions are $u(0,t) - u_x(0,t) = 0$ and $u(1,t) + u_x(1,t) = 0$. The solutions are expected to take longer because the errors introduced at the boundary due to extrapolation will require smaller step sizes in DIFSUB. Because of this only .1 and .01 will be considered for values of EPS. These results are given in Tables 12 and 13, respectively.

t	$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$		$x = \frac{3}{4}$	
	solution	error	solution	error	solution	error	solution	error
.128	.155	.33E-3	.179	-.41E-4	.171	-.44E-3	.095	-.69E-3
.253	.045	-.16E-4	.052	-.14E-3	.050	-.26E-3	.028	-.27E-3
.502	.38E-2	-.21E-4	.44E-2	-.35E-4	.42E-2	-.44E-4	.23E-2	-.36E-4
1.00	.27E-4	-.42E-6	.31E-4	-.57E-6	.30E-4	-.62E-6	.16E-4	-.42E-6
2.00	.13E-8	-.5E-10	.16E-8	-.62E-10	.15E-8	-.63E-10	.83E-9	-.39E-10
4.00	.4E-17	-.2E-18	.4E-17	.9E-19	.4E-17	-.1E-19	.2E-17	.2E-18
8.00	-.5E-20	-.5E-20	.2E-20	.2E-20	-.5E-20	.5E-20	+.9E-21	.9E-21

Table 8. Solution for Example 2, $\Delta x = \frac{1}{256}$, Maximum Error, Using Crank-Nicholson Method

t	$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$		$x = \frac{3}{4}$	# of eqs. evaluated	number of steps	number of eqs./point
	solution error	solution error	solution error	solution error	solution error	solution error	solution error			
.122	.165	.50E-4	.191	.58E-4	.182	.55E-4	.100	.30E-4	12	3
.255	.044	.7E-4	.051	.90E-4	.049	.86E-4	.027	.47E-4	19	3
.508	.37E-2	.46E-4	.42E-2	.53E-4	.41E-2	.51E-4	.22E-2	.28E-4	28	3
.963	.37E-4	-.32E-5	.43E-4	-.36E-5	.41E-4	-.34E-5	.22E-4	-.19E-5	36	3
1.96	-.25E-4	-.25E-4	-.29E-4	-.29E-4	-.27E-4	-.27E-4	-.15E-4	-.15E-4	37	3
3.96	-.11E-4	-.11E-4	-.13E-4	-.13E-4	-.12E-4	-.12E-4	-.66E-5	-.66E-5	39	3
7.96	-.21E-5	-.21E-5	-.25E-5	-.25E-5	-.24E-5	-.24E-5	-.13E-5	-.13E-5	43	1
15.96	-.74E-7	-.74E-7	-.86E-7	-.86E-7	-.82E-7	-.82E-7	-.45E-7	-.45E-7	61	1

Table 9. Solution for Example 2, EPS = .1, Using the System

t	$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$		$x = \frac{3}{4}$		# of eqs. evaluated	number of steps	number of eqs./point
	solution error	solution error	solution error	solution error	solution error	solution error	solution error	solution error			
.122	.165	.5E-4	.191	.58E-4	.182	.55E-4	.100	.30E-4	3780	12	3
.255	.044	.7E-4	.051	.90E-4	.049	.86E-4	.027	.47E-4	5049	19	3
.508	.37E-2	.46E-4	.42E-2	.53E-4	.41E-2	.51E-4	.22E-2	.28E-4	5778	28	3
.963	.37E-4	-.32E-5	.43E-4	-.36E-5	.41E-4	-.34E-5	.22E-4	-.19E-5	7776	36	3
1.96	-.25E-4	-.25E-4	-.29E-4	-.29E-4	-.27E-4	-.27E-4	-.15E-4	-.15E-4	8559	37	3
3.96	-.11E-4	-.11E-4	-.13E-4	-.13E-4	-.12E-5	-.12E-5	-.66E-5	-.66E-5	8667	39	3
7.96	+.24E-4	+.24E-4	+.14E-4	+.14E-4	+.68E-5	+.68E-5	-.30E-5	-.30E-5	9135	43	2
.15.96	.25E-5	.25E-5	.18E-5	.18E-5	-.78E-6	-.78E-6	.11E-6	.11E-6	13842	99	1

Table 10. Solution for Example 2, EPS = .01, Using the System

t	$x = \frac{1}{4}$		$x = \frac{3}{8}$		$x = \frac{1}{2}$		$x = \frac{3}{4}$	# of eqs. evaluated	number of steps	number of eqs./point
	solution error	solution error	solution error	solution error	solution error	solution error	solution error			
.121	.166	.44E-4	.191	.57E-4	.183	.54E-4	.101	.30E-4	4644	12
.254	.045	.78E-4	.052	.40E-4	.046	.86E-4	.040	.70E-4	5967	19
.506	.38E-2	.46E-4	.43E-2	.53E-4	.41E-2	.51E-4	.23E-2	.42E-4	6642	28
.955	.41E-4	-.32E-5	.47E-4	-.37E-5	.45E-4	-.35E-5	.25E-4	-.19E-5	8640	36
1.67	.32E-5	.31E-5	.36E-5	.36E-5	.35E-5	.34E-5	.19E-5	.19E-5	9477	38
3.67	.13E-5	.13E-5	.16E-5	.16E-5	.15E-5	.15E-5	.84E-6	.84E-6	10368	40
7.67	.27E-6	.27E-6	.31E-6	.31E-6	.30E-6	.30E-6	.16E-6	.16E-6	10530	44
15.67	.10E-7	.10E-7	.12E-7	.12E-7	.11E-7	.11E-7	.61E-8	.61E-8	10800	52

Table 11. Solution for Example 2, HPS = .001, Using the System

t	$x = \frac{1}{8}$	$x = \frac{1}{4}$	$x = \frac{3}{8}$	$x = \frac{1}{2}$	# of eqs. evaluated	number of steps	number of eqs./points
.137	.3902	.4194	.4374	.4435	6723	23	3
.313	.2888	.3099	.3228	.3272	7641	25	3
.455	.2264	.2430	.2532	.2566	7695	26	3
.953	.0955	.1026	.1069	.1083	8613	28	3
2.068	.0133	.0142	.0148	.0150	10395	31	3
4.019	.383E-3	.409E-3	.424E-3	.429E-3	11431	35	3
8.070	.709E-6	.295E-6	-.153E-6	-.343E-6	11863	43	3
16.17	.28E-7	.22E-8	-.22E-7	-.33E-7	12727	59	3

Table 12. Solution for Example 3, EPS = .1, Using the System

t	$x = \frac{1}{8}$	$x = \frac{1}{4}$	$x = \frac{3}{8}$	$x = \frac{1}{2}$	# of eqs. evaluated	number of steps	number of eqs./points
.118	.4017	.4331	.4531	.4600	7641	24	3
.240	.3276	.3516	.3663	.3712	9423	27	3
.5315	.1990	.2136	.2225	.2255	11259	31	3
1.037	.0838	.0899	.0937	.0949	11583	37	3
2.036	.0152	.0163	.0170	.0172	12987	48	3
3.959	.565E-3	.607E-3	.632E-3	.640E-3	14067	68	3
7.517	.13E-5	.14E-5	.14E-5	.15E-5	14746	105	1
15.517	-.41E-10	.41E-11	.43E-11	-.18E-10	15048	113	1

Table 13. Solution for Example 3, EPS = .01, Using the System

The independent solution was again obtained by using the Crank-Nicolson method. The program used to solve this system of equations was based on information from Özisik [9]. The program listing is given in Appendix D. The results are given in Table 14.

To obtain a solution using the Crank-Nicholson method requires approximately six seconds. A similar solution requires about 34 seconds using the system.

In examples 2 and 3 above, it was found that results were easier to obtain if the auxiliary algebraic equations at the boundaries were bypassed. This is done by setting up equations of the form

$$v_i = B(u_i, v_i, x_i, t_i)$$

$$r = v_i - B(u_i, v_i, x_i, t_i)$$

where B represents an interpolation polynomial.

In order that the Jacobian will remain invertible, a 1 is inserted in the main diagonal of the equations bypassed. The dependence that interior points would have had on these boundary values, as present in the Jacobian, is now transferred to the interior points used to obtain that value.

Although the error estimate used for the spatial truncation error appears to be accurate enough for the previous examples, the presence of a more accurate one may be necessary in more complicated problems. The system was used to try to integrate a problem where the initial value involved a peak in the center of the interval. Although the system was able to produce values of adequate precision, which would appear to indicate an analog of sufficient accuracy, the error estimates were large enough that the integration would have been rejected. (See Table 15)

t	$x = \frac{1}{8}$	$x = \frac{1}{4}$	$x = \frac{3}{8}$	$x = \frac{1}{2}$
.125	.3974	.4278	.4469	.4534
.25	.3220	.3457	.3601	.3650
.5	.2102	.2256	.2350	.2382
1.0	.0895	.0961	.1000	.1014
2.0	.0162	.0174	.0182	.0184
4.0	.534E-3	.573E-3	.597E-3	.605E-3
8.0	.578E-6	.620E-6	.646E-6	.655E-6
16.0	.67E-12	.73E-12	.76E-12	.77E-12

Table 14. Example 3, Crank-Nicholson Method

step number	t	$x = \frac{1}{8}$		$x = \frac{1}{4}$		$x = \frac{1}{2}$	
		solution	error	solution	error	solution	error
20	.19E-3	.0764920	.49E-7	.1396	.93E-8	1.56568	-.14E-5
40	.45E-3	.0766674	-.22E-7	.140081	.86E-7	1.45653	-.22E-5
60	.65E-3	.0767114	-.17E-5	.140206	.13E-6	1.36786	-.23E-5
80	.88E-3	.0766639	-.76E-5	.140135	.15E-6	1.29436	-.22E-5
100	.11E-2	.076556	-.17E-4	.140051	.13E-6	1.23236	-.21E-5
120	.13E-2	.0764107	-.27E-4	.140106	.95E-7	1.17925	-.19E-5

Table 15. Solution of Example with Peak in Initial Value

A similar situation occurred with the following example:

$$u_t = \frac{\partial}{\partial x} (1+u) \frac{\partial u}{\partial x}, \quad u(0,t) = u(1,t) = 0 \quad u(x,0) = \sin^3(\pi x) \\ x \in [0,1]$$

For use with the system, the equation was rewritten

$$u_t = \left(\frac{\partial u}{\partial x}\right)^2 + (1+u) \frac{\partial^2 u}{\partial x^2}$$

An independent solution was obtained by using a predictor-corrector modification of the Crank-Nicholson method that has been developed by Douglas and Jones [10]. The solution was also obtained for a few values of t using the system with the error control portions bypassed. The results agreed with those obtained with the alternative method. However, the error estimates for the space analogs would have caused the integration to cease. (See Table 16)

The results obtained from the first three examples appear to indicate the feasibility of using the method of lines with the subprogram DIFSUB to produce an automatic program when the asymptotic error equations are integrated along with the original system. The two additional examples seem to provide the possibility of integrating other types of problems including those of a nonlinear nature.

Independent Solution

t	$x = \frac{1}{8}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$
.125	.07123	.12740	.16294	.17507
.25	.01905	.03487	.04526	.04888
.5	.001564	.002887	.003770	.004080

Solution Produced by the System

.124	.07099	.1270	.1624	.1745
.252	.01849	.0338	.0439	.04745
.498	.00158	.00292	.00381	.00413

Table 16. Solution of a Non-Linear Problem, Using the System

LIST OF REFERENCES

- [1] Gear, C. W. "The Automatic Integration of Ordinary Differential Equations," CACM, 14, #3, 1971, pp. 176-179.
- [2] Wendroff, B. THEORETICAL NUMERICAL ANALYSIS, Academic Press, New York, 1966, pp. 1-7.
- [3] Gear, C. W. NUMERICAL INITIAL VALUE PROBLEMS IN ORDINARY DIFFERENTIAL EQUATIONS, Prentice-Hall, Englewood Cliffs, New Jersey, 1971, pp. 155-167.
- [4] Gear, C. W. (private communication)
- [5] Gear, C. W. "Simultaneous Numerical Solution of Differential-Algebraic Equations," IEEE Transactions on Circuit Theory, CT-18, #1, 1971, pp. 89-95.
- [6] Forsythe, G. and Moler, C. COMPUTER SOLUTION OF LINEAR ALGEBRAIC SYSTEMS, Prentice-Hall, Englewood Cliffs, New Jersey, 1967, pp. 68-69.
- [7] Crank, J. and Nicholson, P. "A Practical Method for Numerical Integration of Solutions of Partial Differential Equations of Heat-Conduction Type," Proceedings Cambridge Philos. Society, 43, 1947, p. 50.
- [8] Richtmyer, R. and Morton, K., DIFFERENCE METHODS FOR INITIAL VALUE PROBLEMS, John Wiley and Sons, New York, 2nd Edition, 1967, p. 189 and pp. 198-201.
- [9] Özisik, M. BOUNDARY VALUE PROBLEMS IN HEAT CONDUCTION, International Textbook Company, Scranton, Pennsylvania, 1968, pp. 402-405.
- [10] Douglas, Jr., J. and Jones, Jr., B. Journal Soc. Ind. Appl. Math, 11, 1963, pp. 195-198.

APPENDIX A
THE SYSTEM OF PROGRAMS

```

SUBROUTINE PARDIF(RINT,NCK,DIVER,NPSAVE,PSAVE,EPS,SMAX,SMIN,SX,
1SAVE,NEQ,MAXAR,MAXN,IDER,MSTRT,INP,NUM,MOLD,Y,X,H,HMIN,HMAX,YMAX,
2ERROR,KFLAG,JSTART,MAXDER,K7,HMY,EROR,T,INT,HSAB,PEQ,ERORA)
C
C THE PURPOSE OF THIS PROGRAM IS TO INTEGRATE ONE STEP. TO DO
C THIS IT CREATES A PARTITION, INTEGRATES, AND ATTEMPTS TO CONTROL
C THE ERRORS
C THE VARIABLES ARE
C RINT      BOUNDARY POINTS
C NCK       USED IN XSTP
C DIVER     CONTAINS ERROR ESTIMATE COEFFICIENTS
C NPSAVE    USED IN XSTP
C PSAVE     CONTAINS DECOMPOSED JACOBIAN
C EPS       ERROR PARAMETER - USER
C SMAX,     LIMITS ON SPACE STEP SIZE
C SMIN      STORES Y
C SAVE
C MSTRT,
C MAXAR     LIMITS ON ANALOG TYPE
C           1 - LAGRANGIAN
C           2 - HERMITE
C           3 - TRIMITE
C IDER      DERIVATIVES USED
C Y         ANSWER VECTOR
C X         PARTITION
C HMIN,
C HMAX      LIMITS ON TIME STEP SIZE
C YMAX      USED FOR RELATIVE ERROR WHEN Y IS GREATER THAN 1
C ERROR     TIME ERROR
C KFLAG     RETURN CODE
C HMY       ASYMPTOTIC ERROR VALUE
C EROR      SPACE ERROR
C HSAV      SAVES HMY
C PEQ       JACOBIAN
C ERORA     ESTIMATED ERROR INTRODUCED IN THIS STEP
C
C IMPLICIT REAL * 8(A-H,P-Z)
C DIMENSION ERORA(1)
C DIMENSION KL8(3),RINT(2),NCK(1),DIVER(9,1),NPSAVE(1),PSAVE(1),SX(3
1),NEQ(3),IDER(4,3),INT(2,1),X(1),Y(5,1),YT(3),YTA(3,5),XT(5),
2ERROR(1),EROR(1),YMAX(1),HMY(2,1),GAM(3),SAVE(7,1),HSAB(3,1),
3PEQ(1),MAXM(3)
C COMMON /DVR/ DVY(4),COEF(12,3),ALPHA(2,4,3),DIV(16,5)
C COMMON /CNT/ ICSTPS,NUMDE,NUMSO
C DATA PVAL/.5/
C DATA RVAL,SRVAL/.1D0,.01D0/
C DATA KL8 /9,19,29/
C
C THIS SECTION INITIALIZES COUNTERS AND CHECKS FOR CHANGE CONDITIONS
C
C IF(JSTART.NE.0) GO TO 10
C MAXSV = MAXAR
C ISPSCT = 0
C TPS = .4 * EPS

```

SPS = .1 * EPS	PDF15400
VEPS = DLOG(EPS)	PDF15500
N1=7*MAXN+1	PDF15600
N2=10*MAXN+1	PDF15700
DO 15 IA=1,3	PDF15800
15 MAXM(IA) = MAXN/IA -2	PDF15900
NUMA=MAXN	PDF16000
5 INP = 1	PDF16100
SEPS = EPS	PDF16200
JSTART = 0	PDF16300
KTPSCT = 0	PDF16400
ICPART = 0	PDF16500
KSPSCT = 0	PDF16600
ITUPS = 1	PDF16700
IRECSP = 1	PDF16800
ICK = 0	PDF16900
NCNGE = 0	PDF17000
ITPSC = 0	PDF17100
ICSTPS = 0	PDF17200
NUMDE = 0	PDF17300
NUMSD = 0	PDF17400
T = 0.000	PDF17500
MAXER = MAXSV	PDF17600
GO TO 200	PDF17700
20 IF(NCNGE .EQ. 0) GO TO 255	PDF17800
NCNGE = 0	PDF17900
GO TO 12	PDF18000
10 IF(SEPS .EQ. EPS) GO TO 20	PDF18100
VEPS = DLOG(EPS)	PDF18200
TPS = .4 * EPS	PDF18300
SPS=.1*EPS	PDF18400
12 IF(ICSTPS .LT. 10) GO TO 5	PDF18500
200 CALL XSTP(RINT,NCK,DIVER,NPSAVE(N1),NPSAVE(N2),SPS,SMAX,SMIN, SX	PDF18600
1,KFLAG,NEQ,MAXER,MAXM,IDER,MSTRT,PSAVE,INP,NUMA,MOLD,X,Y,MO)	PDF18700
IF(KFLAG .LT. C .AND. MAXSV .EQ. 1) RETURN	PDF18800
IF(KFLAG .LT. 0) GO TO 299	PDF18900
IF(ICK .EQ. 1 .AND. MO*NEQ(MO) .GT. NUM) GO TO 208	PDF19000
MAXER = MO	PDF19100
	PDF19200
THIS SECTION PUTS THE NEW PARTITION INTO USABLE FORM AND	PDF19300
OBTAINS THE FUNCTIONAL VALUES FOR THIS NEW PARTITION.	PDF19400
	PDF19500
INP=2	PDF19600
J=NEQ(MO)	PDF19700
SAVE(1,1)=RINT(1)	PDF19800
JA=MO*(J+1)+1	PDF19900
SAVE(1,JA)=RINT(2)	PDF20000
JA=MO-1	PDF20100
IF(JA.LE .0) JA=3	PDF20200
IF(JSTART.EQ.0) INT(2,1)=1	PDF20300
IF(JSTART.EQ.0) INT(2,J+2)=J-2	PDF20400
DO 222 IA=1,J	PDF20500
N3= MO+3+(IA-1)+N2-1	PDF20600
IB=NPSAVE(N3)	PDF20700
INT(1,IA+1)=DIVER(MO,IB)	PDF20800

	IAA=1+MO*(IA)	PDF2090
	SAVE(1,IAA)=DIVER(MO+3,IB)	PDF2100
	N3=N3-MO+JA	PDF2110
	IF(IA.EQ.1)NPSAVE(N3)=1	PDF2120
	N3=N3+3	PDF2130
	IF(IA.EQ.1)NPSAVE(N3)=1	PDF2140
	IF(IA.EQ.2) NPSAVE(N3) = 2	PDF2150
	IF(IA.GT.2 .AND. IA.LE.J) NPSAVE(N3) = IA -1	PDF2160
	IF(IA.EQ.J.AND.MO.EQ.1) NPSAVE(N3)=J-2	PDF2170
	IF(JSTART.EQ.0) INT(2,IA+1)=NPSAVE(N3)	PDF2180
222	CONTINUE	PDF2190
	IF(JSTART.NE.0) NPSAVE(N3+3)=NPSAVE(N3)	PDF2200
	NUMA=NEQ(MO)	PDF2210
	IF(JSTART.GE.1) GO TO 300	PDF2220
C		PDF2230
C	NEW VALUES FOR INITIAL PARTITION.	PDF2240
C		PDF2250
	IQQQC=0	PDF2260
	J2=J+2	PDF2270
	DO 250 IA=1,J2	PDF2280
	IAA1 = MO*(IA-1)	PDF2290
	IAA = IAA1 +1	PDF2300
	X(IA)=SAVE(1,IAA)	PDF2310
	CALL INIT(YT,MO,X(IA))	PDF2320
	DO 250 IB=1,MO	PDF2330
	IAA = IAA1 + IB	PDF2340
	HMY(1,IAA) = 0.0	PDF2350
250	Y(1,IAA)=YT(IB)	PDF2360
	GO TO 258	PDF2370
208	MO = MOLO	PDF2380
	J = NUM /MOLO	PDF2390
	NEQ(MO) = J	PDF2400
	GO TO 258	PDF2410
C		PDF2420
C	THIS SECTION FIRST CHECKS TO SEE IF THE POINT WAS IN THE EXISTING	PDF2430
C	PARTITION. IF IT WAS, THE VALUES ARE REUSED. IF NOT, THE NEW	PDF2440
C	VALUES ARE OBTAINED BY INTERPOLATION.	PDF2450
C		PDF2460
300	IW=JSTART+1	PDF2470
	NUM=NUM/MOLO	PDF2480
	J2=J +2	PDF2490
	DO 390 IB=1,J2	PDF2500
	IA=1	PDF2510
	IAA=1+MO*(IB-1)	PDF2520
	XI=SAVE(1,IAA)	PDF2530
310	IF(XI.LE.X(IA)) GO TO 350	PDF2540
	IA=IA+1	PDF2550
	IF(IA.LE.NUM+2) GO TO 310	PDF2560
	KFLAG=-30	PDF2570
	RETURN	PDF2580
10101	KFLAG=-41	PDF2590
	RETURN	PDF2600
350	IF(XI.EQ.X(IA)) GO TO 370	PDF2610
C		PDF2620
C	INTERPOLATION SECTION	PDF2630

C	KG=0	PDF26400
351	IF(IA.LT.NUM+2) IC=INT(2,IA)	PDF26500
	IF(IA.EQ.NUM+2) IC=NUM-1	PDF26600
	IW1 = 1	PDF26700
352	DO 355 IE=1,4	PDF26800
	ID=IC-1+IE	PDF26900
355	XT(IE)=X(ID)	PDF27000
	CALL DERIV(XT,YTA,IDER(1,MO),MO,1,1,KG,MOLD,XI,GAM,XH)	PDF27100
	DO 358 IE = 1,MO	PDF27200
	IAB=IE+MO*(IB-1)	PDF27300
	IF(IW1 .EQ. 1) SAVE(IW+2,IAB) = 0.0	PDF27400
	IF(IW1 .EQ. 1) HSAV(1,IAB) = 0.0	PDF27500
	HSAV(2,IAB) = 0.0	PDF27600
	DO 358 ID = IW1,IW	PDF27700
	SAVE(ID+1,IAB)=0.	PDF27800
	DO 358 IF=1,MOLD	PDF27900
	DO 358 IG=1,4	PDF28000
	IH=IC-1+IG	PDF28100
	IAA=IG+4*(IF-1)	PDF28200
	IAC=IF+MOLD*(IH-1)	PDF28300
	IF(ID .EQ. 1) SAVE(IW+2,IAB) = DMAX1(SAVE(IW+2,IAB),YMAX(IAC))	PDF28400
	IF(ID .EQ. 1) HSAV(1,IAB) = HSAV(1,IAB)+COEF(IAA,IE)*HMY(1,IAC)	PDF28500
	IF(ID .EQ. 2) HSAV(2,IAB) = HSAV(2,IAB)+COEF(IAA,IE)*HMY(2,IAC)	PDF28600
358	SAVE(ID+1,IAB)=SAVE(ID+1,IAB)+COEF(IAA,IE)*Y(ID,IAC)	PDF28700
	IF(IC .GT. 1 .OR. IC .LT. NUM-1) GO TO 390	PDF28800
	IW1 = 2	PDF28900
	IF(IC .EQ. 1) IC = 2	PDF29000
	IF(IC .EQ. NUM-1) IC = IC -1	PDF29100
	GO TO 352	PDF29200
C		PDF29300
C	USING THE OLD PARTITION VALUES	PDF29400
C		PDF29500
370	DO 371 IJ = 1,MO	PDF29600
	IAA=IJ+MO*(IB-1)	PDF29700
	IAB=IJ+MOLD*(IA-1)	PDF29800
	SAVE(IW+2,IAA)=YMAX(IAB)	PDF29900
	HSAV(1,IAA) = HMY(1,IAB)	PDF30000
	HSAV(2,IAA) = HMY(2,IAB)	PDF30100
	DO 371 ID=1,IW	PDF30200
371	SAVE(ID+1,IAA)=Y(ID,IAB)	PDF30300
390	CONTINUE	PDF30400
C		PDF30500
C	UPDATES THE VECTORS FOR THE NEW PARTITION	PDF30600
C		PDF30700
	J2=J+2	PDF30800
	DO 410 IA=1,J2	PDF30900
	N3=N2-1+JA+3*(IA-1)	PDF31000
	INT(2,IA)=NPSAVE(N3)	PDF31100
	IAB=1+MO*(IA-1)	PDF31200
	X(IA)=SAVE(1,IAB)	PDF31300
	DO 410 IC=1,MO	PDF31400
	IAA=IC+MO*(IA-1)	PDF31500
	YMAX(IAA)=SAVE(IW+2,IAA)	PDF31600
	DO 410 IB=1,IW	PDF31700
		PDF31800

	HMY(1,IAA) = HSAV(1,IAA)	PDF31900
	HMY(2,IAA) = HSAV(2,IAA)	PDF32000
410	Y(IB,IAA)=SAVE(IB+1,IAA)	PDF32100
	DO 430 IA = 1,4	PDF32200
430	XT(IA) = X(IA+1)	PDF32300
	XI = X(1)	PDF32400
	CALL DERIV(XT,YTA,IDER(1,MO),1,1,1,0,1,XI,GAM,XH)	PDF32500
	DO 435 IA = 1,MO	PDF32600
	DO 435 IB = 2,IW	PDF32700
	Y(IB,IA) =0.0	PDF32800
	IF(IB .EQ. 2) HMY(2,IA) = 0.0	PDF32900
	DO 435 IF = 1,4	PDF33000
	IE = IA + MO *(IF-1)	PDF33100
	Y(IB,IA) = Y(IB,IA) + COEF(IF,1)*Y(IB,IE)	PDF33200
	IF(IB .EQ. 2) HMY(2,IA) = HMY(2,IA) + COEF(IF,1)*HMY(2,IE)	PDF33300
435	CONTINUE	PDF33400
	DO 440 IA = 1,4	PDF33500
440	XT(IA) = X(J2 - IA)	PDF33600
	XI = X(J2)	PDF33700
	CALL DERIV(XT,YTA,IDER(1,MO),1,1,1,0,1,XI,GAM,XH)	PDF33800
	DO 445 IA = 1,MO	PDF33900
	DO 445 IB = 2,IW	PDF34000
	IG = J2 - MO + IA	PDF34100
	Y(IB,IG) = 0.0	PDF34200
	IF(IB .EQ. 2) HMY(2,IG) = 0.0	PDF34300
	DO 445 IF = 1,4	PDF34400
	IE = IA + MO * (J2 - IF -1)	PDF34500
	Y(IB,IG) = Y(IB,IG) + COEF(IF,1) * Y(IB,IE)	PDF34600
	IF(IB .EQ. 2) HMY(2,IG) = HMY(2,IG)+ COEF(IF,1)*HMY(2,IE)	PDF34700
445	CONTINUE	PDF34800
	IDOUB=IW	PDF34900
	IQQQC=1	PDF35000
C		PDF35100
C	CALCULATES THE COEFFICIENTS FOR THE ERROR ESTIMATING INTERPOLATING	PDF35200
C	POLYNOMIAL	PDF35300
C		PDF35400
258	DO 262 IA=1,J	PDF35500
	ID=MO*IA+1	PDF35600
	XI=X(IA+1)	PDF35700
	DO 260 IB=1,5	PDF35800
	IC=INT(2,IA+1)-1+IB	PDF35900
	XT(IB)=X(IC)	PDF36000
260	CONTINUE	PDF36100
	IAA=INT(1,IA+1)+IA	PDF36200
	XH=XI-X(IAA+1)	PDF36300
	CALL DERIV(XT,YTA,IDER(1,MO),MO,1,0,1,MOLD,XI,GAM,XH)	PDF36400
	DO 262 IB=1,MO	PDF36500
	IAA1 = 9*(IB-1)	PDF36600
	DO 262 IC=1,9	PDF36700
	IAA = IAA1 + IC	PDF36800
262	DIVER(IAA,ID)=COEF(IC,IB)	PDF36900
	K7=KL8(MO)	PDF37000
	N=(NEQ(MO)+2)*MO	PDF37100
C		PDF37200
C	INTEGRATES AND DETERMINES THE MAGNITUDE OF THE ERROR	PDF37300

C			PDF37400
255	CALLDIFSUBIN,T,Y,SAVE,H,HMIN,HMAX,TPS,YMAX,ERROR,KFLAG,JSTART,		PDF37500
	1MAXDER,PSAVE,K7,HMY,X,DIVER,INT,IDER(1,MO),EROR,MO,IDOUB,IQQQC,		PDF37600
	2HSAV,PEQ,ERORA,ITPSC)		PDF37700
	ICSTPS = ICSTPS + 1		PDF37800
	ICK = 0		PDF37900
	IQQQC=0		PDF38000
	NUM=NEQ(MO)*MO		PDF38100
	MOLD=MO		PDF38200
	IW=JSTART+1		PDF38300
	IF(KFLAG.LE.-3) GO TO 290		PDF38400
	IF(KFLAG.LE.0) RETURN		PDF38500
	NUM1=NUM+MO		PDF38600
	NUM2=MO+1		PDF38700
	RMAX = 0.0		PDF38800
	DMAX = 0.0		PDF38900
	DO 270 IA=NUM2,NUM1		PDF39000
	DBSEA = DABS(E RORA(IA))		PDF39100
	IF(DBSEA.GT.RMAX) RMAX = DBSEA		PDF39200
	IF(ICPART.LT.3) GO TO 270		PDF39300
	DMAXT = DABS(EROR(IA))		PDF39400
	IF(DMAXT.GT.DMAX) DMAX = DMAXT		PDF39500
270	CONTINUE		PDF39600
C			PDF39700
C	CHECKS ON ACTIONS BASED ON ERRORS AND COUNTERS		PDF39800
C			PDF39900
	IF(RMAX.GE.EPS) GO TO 290		PDF40000
C			PDF40100
C	ERROR IS SMALL ENOUGH		PDF40200
			PDF40300
	SEPS = EPS		PDF40400
	IF(RMAX.GE.RVAL*EPS.OR.(ICSTPS-IRECSP).LT.10) RETURN		PDF40500
	ICPART = ICPART + 1		PDF40600
	IRECSP = ICSTPS		PDF40700
	IF(ICPART.LE.3) GO TO 275		PDF40800
	ICPART = 0		PDF40900
	IF(DMAX.LT.SRVAL*EPS) GO TO 292		PDF41000
275	IF (ITUPS.GT.3) GO TO 288		PDF41100
C			PDF41200
C	CHANGES TIME PARAMETER		PDF41300
C			PDF41400
	VTPS = DLOG (TPS)		PDF41500
	VTPS = (VTPS + VEPS) / 2.		PDF41600
	TTPS = DEXP (VTPS)		PDF41700
	TPS = DMIN1 (TTPS,TPS+.5D)*(EPS-RMAX))		PDF41800
	ITUPS = ITUPS + 1		PDF41900
	RETURN		PDF42000
288	IF(RMAX.GT.SRVAL*EPS) RETURN		PDF42100
C			PDF42200
C	CHANGES SPACE PARAMETER		PDF42300
C			PDF42400
291	IF(SX(MO)*2.0DC.GE.SMAX.AND.MO.EQ.MSTRT) RETURN		PDF42500
	ICK = 1		PDF42600
	TSPS = 2. * SPS		PDF42700
	SPS = DMIN1(TSPS,.5 * EPS)		PDF42800

	TPS = .6 * EPS -SPS	PDF42900
292	IF(SX(MO)*2.000 .GE. SMAX .AND. MO .EQ. MSTRT) RETURN	PDF43000
	ITUPS = 1	PDF43100
	ICPART = 0	PDF43200
	NCNGE = 1	PDF43300
	RETURN	PDF43400
C		PDF43500
C	ERROR TOO LARGE	PDF43600
C		PDF43700
290	ITUPS = 1	PDF43800
	ICPART = 0	PDF43900
	ICSTPS = ICSTPS - 1	PDF44000
	IRECSP = ICSTPS	PDF44100
	IF(ICSTPS .LT. 10) GO TO 298	PDF44200
	IF(KFLAG .LE. -3) GO TO 293	PDF44300
	IF(TPS .LT. .00001 .OR. KTPSCT .GE. 5) GO TO 296	PDF44400
	GO TO 294	PDF44500
296	KSPSCT = KSPSCT + 1	PDF44600
	KTPSCT = 1	PDF44700
	IF(KSPSCT .LE. 10) GO TO 293	PDF44800
	KFLAG = -42	PDF44900
	RETURN	PDF45000
C		PDF45100
C	IF THE NUMBER OF STEPS IS SMALL ENOUGH - RESTARTS WITH SMALLER	PDF45200
C	PARAMETERS	PDF45300
298	SPS = .1 * SPS	PDF45400
	TPS = .1 * TPS	PDF45500
	NCNGE = 1	PDF45600
	ISPSCT = ISPSCT + 1	PDF45700
	IF(ISPSCT .LT. 4) GO TO 10	PDF45800
	KFLAG = -35	PDF45900
	IF(MAXSV .EQ. 1) RETURN	PDF46000
299	MAXSV = 1	PDF46100
	IF(ICSTPS .GT. 10) RETURN	PDF46200
	GO TO 2	PDF46300
C		PDF46400
C	DECREASES SPACE PARAMETER	PDF46500
C		PDF46600
293	IF(TPS .LT. .00001) GO TO 295	PDF46700
	TPS = .000001	PDF46800
	JSTART = -1	PDF46900
	GO TO 255	PDF47000
295	SPS = PVAL * SPS	PDF47100
	GO TO 400	PDF47200
C		PDF47300
C	DECREASES TIME PARAMETER	PDF47400
C		PDF47500
294	VTPS = DLOG(TPS)	PDF47600
	VTPS = (VTPS + DLOG(1.0-6))/2.00	PDF47700
	KTPSCT = KTPSCT + 1	PDF47800
	TPS = DEXP (VTPS)	PDF47900
	ITPSC = 1	PDF48000
	JSTART = -1	PDF48100
	GO TO 255	PDF48200
C		PDF48300

C	OBTAINS THE OLD PARTITION SO THAT THE NEW PARTITION MAY BE	PDF48400
C	CALCULATED	PDF48500
C		PDF48600
400	CONTINUE	PDF48700
	DO 405 KF1=1,N	PDF48800
	HMY(1,KF1) = HSAV(1,KF1)	PDF48900
	HMY(2,KF1) = HSAV(2,KF1)	PDF49000
	DO 405 KF2=1,IW	PDF49100
405	Y(KF2,KF1)=SAVE(KF2,KF1)	PDF49200
	GO TO 200	PDF49300
	END	PDF49400

SUBROUTINE XSTP(RINT,NCK,DPOS,NPT,NPTER,EROR,HMAX,HMIN,SX,K,NEQ,		STP10000
IMAXO,MAXM,IDER,MSTRT,YTEMP,INP,NUM,MOLD,X,Y,MONW)		STP10100
C		STP10200
C		STP10300
C	THE PURPOSE OF THIS SUBPROGRAM IS TO PROVIDE AN INITIAL	STP10400
C	PARTITION OR MODIFY AN EXISTING ONE	STP10500
C	THE VARIABLES IN THIS SUBPROGRAM ARE	STP10600
C	RINT BOUNDARY POINTS	STP10700
C	NCK AN ARRAY +1 INDICATES POSITION SATISFIES	STP10800
C	ERROR CONDITION; -1 INDICATES IT DOES NOT	STP10900
C	DPOS CONTAINS PARTITION POINTS ALONG WITH ITS	STP11000
C	ANALOG NEIGHBORS	STP11100
C	NPT POINTER ARRAY FOR OLD & NEW PASSES THROUGH	STP11200
C	THE INTERVAL	STP11300
C	NPTER OUTPUT POINTER FOR ARRAY	STP11400
C	EROR SPATIAL ERROR PARAMETER	STP11500
C	HMAX LARGEST ALLOWABLE STEP SIZE	STP11600
C	HMIN SMALLEST ALLOWABLE STEP SIZE	STP11700
C	SX CONTAINS THE SMALLEST STEPS USED	STP11800
C	K RETURN CODE	STP11900
C	MAXO MAXIMUM ORDER ALLOWED	STP12000
C	MAXM MAXIMUM NUMBER OF POINTS IN EACH ORDER	STP12100
C	IDER CONTAINS CODE FOR DERIVATIVES USED	STP12200
C	MSTRT LOWEST ORDER ALLOWED	STP12300
C	YTEMP TEMPORARY STORAGE OF PARTITION FUNCTION VALUES	STP12400
C	INP 1 - INITIAL PARTITION; 2 - MODIFY	STP12500
C	NUM NUMBER OF POINTS IN OLD PARTITION	STP12600
C	MOLD OLD ORDER	STP12700
C	X OLD PARTITION	STP12800
C	Y OLD FUNCTION VALUES	STP12900
C	MONW NEW ORDER	STP13000
C	AF TEMPORARY STORAGE FOR TIME DERIVATIVE	STP13100
C	AD TEMPORARY STORAGE FOR ERROR ESTIMATE IN AF	STP13200
C	AVA RELATIVE POSITION OF POINTS USED IN INITIAL	STP13300
C	PARTITION CALCULATION	STP13400
C	AA TEMPORARILY STORES SPACE DERIVATIVE	STP13500
C	AE TEMPORARILY STORES PERTURBATION OF AA	STP13600
C	YT,XT STORES VALUES FOR DERIV	STP13700
C	L RECORDS POSITIONS FOR POINTS IN ERROR EXTIMATE	STP13800

```

C      FOR DVY,ALPHA,DIV,COEF,GAM      SEE DERIV      STP13900
C
      IMPLICIT REAL * 8 (A-H,Q-Z)      STP14000
      DIMENSION RINT(2),NCK(1),DPOS(3,3,1),NPT(2,1),NPTER(3,1),A(3,2,5), STP14100
      1AF(3),AD(3),GAM(3),AVA(5),AA(4),AE(4),HQ(2),HSQ(2),IDER(4,3), STP14200
      2YT(3,5),XT(5),YTEMP(3,1),L(5) STP14300
      DIMENSION SX(3),NEQ(3) STP14400
      DIMENSION ALFA(2),X(1),Y(5,MOLD,1),MAXM(3) STP14500
      DIMENSION IWD1(3,4),RWD(3,2),WVAR(3) STP14600
      COMMON /CNT/ ISTEP,NUMDE,NUMSO STP14700
      COMMON /DVR/ DVY(4),COEF(4,3,3),ALPHA(2,4,3),DIV(16,5) STP14800
      DATA IWD1,RWD/8,9,18,18,26,28,2,2,5,5,7,8,2,,6,,10,,3,,9,,15./ STP14900
      DATA ALFA/1.00,,.9500/ STP15000
      DATA AVA/-1.,-.50000000,0.0,,.250000000,1. / STP15100
      AT = ALFA(INP) STP15200
      RATIO = .10000000 STP15300
      IF(INP.NE. 2 .OR. NUMSO .LE.40) GO TO 10 STP15400
      RATIO = FLOAT(NUMDE)/FLOAT(NUMSO) STP15500
10    CONTINUE STP15600
      DHMIN=HMIN/2. STP15700
      K=1 STP15800
C      STP15900
C      CHECK INPUT PARAMETERS STP16000
C      STP16100
      IF(MSTRT .GT. MAXO .OR. MSTRT .LE. 0) GO TO 2000 STP16200
      MO = MSTRT STP16300
      RNT=RINT(2)-RINT(1) STP16400
      IF(RNT .LE. 0.00) GO TO 2010 STP16500
      IF(MAXO .GT. 3 .OR. MAXO .LT. 1) GO TO 2000 STP16600
C      STP16700
C      INITILIZE PARTITION VALUES FOR NEW ORDER STP16800
C      STP16900
20    MAXN=MAXM(MO) STP17000
      DO 30 I=1,MAXN STP17100
30    NCK(I)=-1 STP17200
      H=RNT/2. STP17300
      DPOS(MO,2,1)=RINT(1)+H STP17400
      H=H/2. STP17500
      N=3 STP17600
      SX(MO)=H STP17700
      NEQ(MO)=3 STP17800
      NPT(1,1)=2 STP17900
      NPT(1,2)=1 STP18000
      NPT(1,3)=3 STP18100
      DPOS(MO,2,MAXN+1)=RINT(1) STP18200
      DPOS(MO,2,MAXN+2)=RINT(2) STP18300
      DPOS(MO,1,2)=RINT(1) STP18400
      DPOS(MO,2,2)=RINT(1)+H STP18500
      DPOS(MO,1,1)=DPOS(MO,2,2) STP18600
      DPOS(MO,3,2)=DPOS(MO,2,1) STP18700
      DPOS(MO,1,3)=DPOS(MO,2,1) STP18800
      DPOS(MO,2,3)=RINT(2)-H STP18900
      DPOS(MO,3,3)=RINT(2) STP19000
      DPOS(MO,3,1)=DPOS(MO,2,3) STP19100
      IF(INP .NE. 1) GO TO 1800 STP19200
      STP19300

```

C			STP19400
C		INITILIZES ITEMS FOR THIS PASS OF INTERVAL	STP19500
C			STP19600
100	H=H/2.		STP19700
	ILD=1		STP19800
	INW=2		STP19900
110	JLD=1		STP20000
	JNW=1		STP20100
	NMCK=0		STP20200
	HH=H		STP20300
	HHP=HH*.94159265		STP20400
	HQ(1)=HH		STP20500
	HQ(2)=HHP		STP20600
	HSQ(1)=HH*HH		STP20700
	HSQ(2)=HHP*HHP		STP20800
C			STP20900
C		SETS UP THE VALUES FOR ADDITIONAL POINTS	STP21000
C			STP21100
120	I=NPT(ILD,JLD)		STP21200
	IF(NCK(I).GT.0) GO TO 600		STP21300
	FWD=DPOS(MO,2,I)+H		STP21400
	BWD=DPOS(MO,2,I)-H		STP21500
C			STP21600
C		CHECKS TO SEE IF THE POINT IS ALREADY IN THE PARTITION	STP21700
C			STP21800
	KA=-1		STP21900
	DO 130 J=1,N		STP22000
	IF(DABS(BWD-DPOS(MO,2,J)).LT.DHMIN) GO TO 220		STP22100
	IF(DABS(BWD-H-DPOS(MO,2,J)).LT.DHMIN) KA=1		STP22200
130	CONTINUE		STP22300
	IF(DABS(BWD-H-RINT(1)) .LT.DHMIN) KA=1		STP22400
	IF(KA .LE. 0) GO TO 2020		STP22500
C			STP22600
C		CHECKS TO SEE IF MAXIMUM NUMBER OF EQUATIONS IS EXCEEDED	STP22700
C			STP22800
	IF(N+1 .GT. MAXN) GO TO 2030		STP22900
	N=N+1		STP23000
C			STP23100
C		SETS POINT LEFT OF PRESENT PARTITION POINT INTO PARTITION	STP23200
C		AND PREPARES TO CHECK IT	STP23300
C			STP23400
	DPOS(MO,2,N)=BWD		STP23500
	NPT(INW,JNW)=N		STP23600
	JNW=JNW+1		STP23700
	DPOS(MO,1,N)=DPOS(MO,1,I)		STP23800
	JC=N		STP23900
	DPOS(MO,3,N)=DPOS(MO,2,I)		STP24000
	JA=1		STP24100
	IF(H.GT.HMAX) GO TO 560		STP24200
	ERAR = EROR		STP24300
	IF(INP .EQ. 1) GO TO 300		STP24400
	KLA=4		STP24500
	XI=DPOS(MO,2,N)		STP24600
	GO TO 1850		STP24700
210	L(4)=I		STP24800

	L(2)= NPT(INW,JNW-2)	STP24900
	L(1)=NPT(INW,JNW-3)	STP25000
	L(5)= NPT(ILD,JLD+1)	STP25100
	IF(JLD.GE.NEQ(MO)) L(5)=MAXN+2	STP25200
	IF(JLD.LE.2) L(1)=MAXN+1	STP25300
	IF(JLD .GT. 1) GO TO 400	STP25400
	L(2)=MAXN+1	STP25500
	L(1)=NPT(ILD,JLD+1)	STP25600
	L(5)=NPT(ILD,JLD+2)	STP25700
	GO TO 400	STP25800
C		STP25900
C	CHECKS THE PRESENT PARTITION POINT AND PUTS POINT TO RIGHT	STP26000
C	INTO THE PARTITION	STP26100
C		STP26200
220	NPT(INW,JNW)=I	STP26300
	JNW=JNW+1	STP26400
	REC=DPOS(MO,3,I)	STP26500
	DPOS(MO,1,I)=BWD	STP26600
	DPOS(MO,3,I)=FWD	STP26700
	IF(N+1.GT.MAXN) GO TO 2030	STP26800
	N=N+1	STP26900
	DPOS(MO,2,N)=FWD	STP27000
	NPT(INW,JNW)=N	STP27100
	JNW=JNW+1	STP27200
	DPOS(MO,1,N)=DPOS(MO,2,I)	STP27300
	DPOS(MO,3,N)=REC	STP27400
	JC=N	STP27500
	JA = 3	STP27600
	ERAR = EROR	STP27700
	IF(INP .EQ. 1) GO TO 300	STP27800
	KLA=5	STP27900
	XI=DPOS(MO,2,N)	STP28000
	GO TO 1850	STP28100
230	L(1)=NPT(INW,JNW-3)	STP28200
	L(2)=NPT(INW,JNW-2)	STP28300
	L(4)=NPT(ILD,JLD+1)	STP28400
	L(5)=NPT(ILD,JLD+2)	STP28500
	IF(JLD.GE.NEQ(MO)-1) L(5)=MAXN+2	STP28600
	IF(JLD.LT.NEQ(MO)) GO TO 400	STP28700
	L(4)=MAXN+2	STP28800
	L(5)=NPT(INW,JNW-4)	STP28900
	GO TO 400	STP29000
C		STP29100
C	CHECKS THE POINT TO THE RIGHT OF THE OLD PARTITION POINT.	STP29200
C		STP29300
240	JA=2	STP29400
	JC=I	STP29500
	ERAR = EROR	STP29600
	IF(INP.EQ.1) GO TO 300	STP29700
	XI=DPOS(MO,2,JC)	STP29800
	L(5)= NPT(ILD,JLD+1)	STP29900
	L(2)=NPT(INW,JNW-3)	STP30000
	L(4)=NPT(INW,JNW-1)	STP30100
	L(1)=NPT(INW,JNW-4)	STP30200
	IF(JLD.EQ.1) L(1)=MAXN+1	STP30300

```

      IF(JLD.GE.NEQ(MO)) L(5)=MAXN+2
      GO TO 400
C
C FINDS FUNCTIONAL VALUES FOR INITIAL PARTITION
C
300 DO 310 JB=1,5
      V1 = DPOS(MO,2,JC)+AVA(JB)*HH
      IF(DABS(V1-RINT(1)).LT. 1.1*HH .OR. DABS(V1-RINT(2)) .LT.
1 1.1*HH) ERAR = .5 * EROR
      CALL INIT(A(1,1,JB),MO,V1)
310 CALL INIT(A(1,2,JB),MO,DPOS(MO,2,JC)+AVA(JB)*HHP)
      GO TO (320,350,380),MO
C
C EVALUATES USING LAGRANGIAN - INITIAL
C
320 JB = 0
330 JB = JB + 1
      IF(IDER(1,1).EQ.0) GO TO 340
      AA(1)=(A(1,JB,5)-A(1,JB,1))/HQ(JB)
      AC=(2.*A(1,JB,5)-12.000000*A(1,JB,3)+16.*A(1,JB,2)-6.000000*A(1,JB,1))/(HQ(JB)*HSQ(JB))
      AE(1)=AA(1)+AC/6.000000 *HSQ(JB)
340 IF(IDER(2,1).EQ.0) GO TO 510
      AA(2)=(A(1,JB,5)-2.*A(1,JB,3)+A(1,JB,1))/HSQ(JB)
      AC=(10.666666666666667*A(1,JB,5)-136.5333333333333*A(1,JB,4)+192.000000*A(1,JB,3)-85.3333333333333*A(1,JB,2)+19.200000*A(1,JB,1))/(HSQ(JB)*HSQ(JB))
      AE(2)=AA(2)+AC/12.000000 *HSQ(JB)
      GO TO 510
C
C EVALUATES USING HERMITE - INITIAL
C
350 JB = 0
360 JB = JB + 1
      AA(1)=A(2,JB,3)
      AE(1)=AA(1)
      IF(IDER(2,2).EQ.0) GO TO 370
      AA(2)= 2.*(A(1,JB,5)-2.*A(1,JB,3)+A(1,JB,1))/HSQ(JB)-(A(2,JB,5)-A(2,JB,1))/(2.*HQ(JB))
      AC= (-613.3333333333333*A(1,JB,5)-8640.0000*A(1,JB,3)+8533.33333333333*A(1,JB,2)+720.00000*A(1,JB,1))/HSQ(JB)+(160.00000*A(2,JB,5)+280.00000*A(2,JB,3)+2560.0000*A(2,JB,2))/HQ(JB)
      AE(2)=AA(2)+AC/320.00000
370 IF(IDER(3,2).EQ.0) GO TO 510
      AA(3)=((A(1,JB,5)-A(1,JB,1))*7.5000000/HQ(JB)-1.5000000*(A(2,JB,5)-A(2,JB,1)+8.*A(2,JB,3)+A(2,JB,1)))/HSQ(JB)
      AC=((35280.000*A(1,JB,1)+47786.66666666667*A(1,JB,2)-80640.000*A(1,JB,3)-2426.666666666667*A(1,JB,5))/HQ(JB)+(5040.0000*A(2,JB,1)+3584.0000*A(2,JB,2)+20160.000*A(2,JB,3)+560.00000*A(2,JB,5)))/HSQ(JB)
      AE(3) = AA(3) + AC/840.000000
      GO TO 510
C
C EVALUATES USING TRIMITE - INITIAL
C
380 JB = 0

```

```

STP30400
STP30500
STP30600
STP30700
STP30800
STP30900
STP31000
STP31100
STP31200
STP31300
STP31400
STP31500
STP31600
STP31700
STP31800
STP31900
STP32000
STP32100
STP32200
STP32300
STP32400
STP32500
STP32600
STP32700
STP32800
STP32900
STP33000
STP33100
STP33200
STP33300
STP33400
STP33500
STP33600
STP33700
STP33800
STP33900
STP34000
STP34100
STP34200
STP34300
STP34400
STP34500
STP34600
STP34700
STP34800
STP34900
STP35000
STP35100
STP35200
STP35300
STP35400
STP35500
STP35600
STP35700
STP35800

```

```

390  JB = JB + 1                                STP3590
      AA(1)=A(2,JB,3)                          STP3600
      AE(1)=AA(1)                              STP3610
      AA(2)=A(3,JB,3)                          STP3620
      AE(2)=AA(2)                              STP3630
      IF (IDER(3,3).EQ.0) GO TO 395             STP3640
      AA(3)=(13.125000/HSQ(JB))*(A(1,JB,5)-A(1,JB,1))-1./HQ(JB)*(4.125000STP3650
10*A(2,JB,5)+18.000000*A(2,JB,3)+4.1250000*A(2,JB,1))+.37500000*(A(STP3660
23,JB,5)-A(3,JB,1))/HQ(JB)                    STP3670
      AC=((64411200.*A(1,JB,1)-263782400.*A(1,JB,2)+199584000.*A(1,JB,3)STP3680
1-212800.00*A(1,JB,5))/HSQ(JB)+(10659600.*A(2,JB,1)-34406400.*A(2,JB,2)STP3690
2B,2)-43545600.*A(2,JB,3)+25200.000*A(2,JB,5))/HQ(JB)+(498960.00*A(STP3700
33,JB,1)-9461760.0*A(3,JB,2)+3265920.0*A(3,JB,3)+1680.0000*A(3,JB,5)STP3710
4))/HSQ(JB)                                    STP3720
      AE(3)=AA(3)+AC/60480.000                 STP3730
395  IF (IDER(4,3).EQ.0) GO TO 510             STP3740
      AA(4)=(72.000000/HSQ(JB))*(A(1,JB,5)-2.*A(1,JB,3)+A(1,JB,1))-19.500STP3750
10000/HQ(JB)                                    STP3760
1*(A(2,JB,5)-A(2,JB,1))+1.5000000*(A(3,JB,5)-24.000000*A(3,JB,3)+A(STP3770
33,JB,1))/HSQ(JB)                             STP3780
      AC=((1146880000.*A(1,JB,2)-152409600.*A(1,JB,1)-1001548800.*A(1,JBSTP3790
1,3)+7078400.0*A(1,JB,5))/HSQ(JB)+(206438400.*A(2,JB,2)-22680000.*ASTP3800
3(2,JB,1)+232243200.*A(2,JB,3)-2049600.0*A(2,JB,5))/HQ(JB)+(3440640STP3810
30.*A(3,JB,2)-907200.00*A(3,JB,1)-21772800.*A(3,JB,3)+168000.00*A(3STP3820
4,JB,5))/HSQ(JB)                             STP3830
      AE(4)=AA(4)+AC/151200.00                STP3840
      GO TO 510                                STP3850
C                                           STP3860
C  PREPARE FOR EVALUATION OF THE DERIVATIVE STP3870
C                                           STP3880
400  L(3)=JC                                    STP3890
      DO 410 IB=1,5                            STP3900
      LA=L(IB)                                STP3910
      XT(IB)=DPOS(MO,2,LA)                    STP3920
      DO 410 IA=1,MO                          STP3930
410  YT(IA,IB)=YTEMP(IA,LA)                  STP3940
      CALL DERIV(XT,YT,IDER(1,MO),MO,0,0,0 ,MOLD,XI,GAM,HQ(1)) STP3950
      GO TO (420,440,460),MO                 STP3960
C                                           STP3970
C  EVALUATES USING LAGRANGIAN - MODIFYING STP3980
C                                           STP3990
420  IF (IDER(1,1).EQ.0) GO TO 430            STP4000
      AA(1)=(YT(1,4)-YT(1,2))/(2.0*HQ(1))    STP4010
      AE(1)=AA(1)+DVY(1)                     STP4020
430  IF (IDER(2,1).EQ.0) GO TO 500           STP4030
      AA(2)=(YT(1,4)-2.*YT(1,3)+YT(1,2))/HSQ(1) STP4040
      AE(2)=AA(2)+DVY(2)                     STP4050
      GO TO 500                               STP4060
C                                           STP4070
C  EVALUATES USING HERMITE - MODIFYING STP4080
C                                           STP4090
440  AA(1)=YT(2,3)                           STP4100
      AE(1)=AA(1)                             STP4110
      IF (IDER(2,2).EQ.0) GO TO 450           STP4120
      AA(2)= 2.*(YT(1,4)-2.*YT(1,3)+YT(1,2))/HSQ(1)-(YT(2,4)-YT(2, STP4130

```

```

12)) / (2.0 * HQ(1))
AE(2) = AA(2) + DVY(2)
450 IF (IDER(3,2).EQ.0) GO TO 500
AA(3) = ((YT(1,4) - YT(1,2)) * 7.50000000 / HQ(1) - 1.5000000 * (YT(2,4) + 8
1. * YT(2,3) + YT(2,2))) / HSQ(1)
AE(3) = AA(3) + DVY(3)
GO TO 500

C
C EVALUATES USING TRIMITE - MODIFYING
C
460 AA(1) = YT(2,3)
AE(1) = AA(1)
AA(2) = YT(3,3)
AE(2) = AA(2)
IF (IDER(3,3).EQ.0) GO TO 470
AA(3) = (13.125000 / HSQ(1) * (YT(1,4) - YT(1,2)) - (4.1250000 * (YT(2,4) +
2YT(2,2)) + 18.000000 * YT(2,3))) /
1HQ(1) + .37500000 * (YT(3,4) - YT(3,2))) / HQ(1)
AE(3) = AA(3) + DVY(3)
470 IF (IDER(4,3).EQ.0) GO TO 500
AA(4) = (72.000000 / HSQ(1) * (YT(1,4) - 2. * YT(1,3) + YT(1,2)) - 19.500000 /
1HQ(1) * (YT(2,4) - YT(2,2)) + 1.5000000 * (YT(3,4) - 24.000000 * YT(3,3) + Y
2T(3,2))) / HSQ(1)
AE(4) = AA(4) + DVY(4)

C
C MAKES USE OF THE USER'S EQUATION
C
500 CALL UFUN(MO,DPOS(MO,2,JC),YT(1,3),T,AA,AD,1)
CALL UFUN(MO,DPOS(MO,2,JC),YT(1,3),T,AE,AF,1)
GO TO 520
510 CALL UFUN(MO,DPOS(MO,2,JC),A(1,JB,3),O.,AA,AD,1)
CALL UFUN(MO,DPOS(MO,2,JC),A(1,JB,3),O.,AE,AF,1)

C
C CHECKS THE ERROR
C
520 DO 540 IL = 1,MO
AF(IL) = AF(IL) - AD(IL)
IF (DABS(AD(IL)) .GT. 1.00) GO TO 530
AD(IL) = AD(IL) / AF(IL)
530 IF (DABS(AF(IL)) .GT. ERAR) GO TO 560
540 CONTINUE
IF (INP .EQ. 2 .OR. JB .GE. 2) GO TO 550
GO TO (330,360,390),MO

C
C INDICATES THAT THE ERROR IS ALL RIGHT, SO THAT THE POINT IS
C IGNORED ON FUTURE PASSES
C
550 NCK(JC) = 1
NMCK = NMCK + 1
560 GO TO (220,610,240),JA

C
C CHECKS FOR COMPLETION OF PASSES. IF NOT DONE, PREPARES
C FOR NEW PASS
C
600 NPT(INW,JNW)=I

```

STP41400
 STP41500
 STP41600
 STP41700
 STP41800
 STP41900
 STP42000
 STP42100
 STP42200
 STP42300
 STP42400
 STP42500
 STP42600
 STP42700
 STP42800
 STP42900
 STP43000
 STP43100
 STP43200
 STP43300
 STP43400
 STP43500
 STP43600
 STP43700
 STP43800
 STP43900
 STP44000
 STP44100
 STP44200
 STP44300
 STP44400
 STP44500
 STP44600
 STP44700
 STP44800
 STP44900
 STP45000
 STP45100
 STP45200
 STP45300
 STP45400
 STP45500
 STP45600
 STP45700
 STP45800
 STP45900
 STP46000
 STP46100
 STP46200
 STP46300
 STP46400
 STP46500
 STP46600
 STP46700
 STP46800

	JNW=JNW+1	STP46900
	NMCK=NMCK+1	STP47000
610	IF(JLD.GE.NEQ(MO)) GO TO 620	STP47100
	JLD=JLD+1	STP47200
	GO TO 120	STP47300
620	IF(NMCK.GE.N) GO TO 790	STP47400
	ILD=3-ILD	STP47500
	INW=3-INW	STP47600
	NEQ(MO)=N	STP47700
	H=H/2.	STP47800
	IF(H.LT.HMIN) GO TO 2090	STP47900
	SX(MO)=H	STP48000
	GO TO 110	STP48100
C		STP48200
C	PREPARE THE WORK ESTIMATE	STP48300
C		STP48400
700	IF(NEQ(MO).GT.0) GO TO 710	STP48500
	WVAR(MO) = 1.0050	STP48600
	GO TO 720	STP48700
710	WVTP = IDER(MO,MO)*IWD1(MO,1) + IDER(MO+1,MO)*IWD1(MO,2)	STP48800
	TMO = 2*MO	STP48900
	WVTP = WVTP/TMO	STP49000
	WVTR = IDER(MO,MO)*IWD1(MO,3) + IDER(MO+1,MO)*IWD1(MO,4)	STP49100
	RN = N	STP49200
	WVAR(MO) = RN*(WVTP + RATIO*(WVTR*RN+RWD(MO,1)) + RWD(MO,2))	STP49300
720	IF(MO.EQ.MAXO) GO TO 730	STP49400
	MO = MO + 1	STP49500
	K=1	STP49600
	GO TO 20	STP49700
C		STP49800
C	DECIDE WHICH ORDER TO USE	STP49900
C		STP50000
730	MO = MSTRT	STP50100
	IF(MSTRT.EQ.MAXO) GO TO 740	STP50200
	IF(WVAR(MO)*AT.GT.WVAR(MO+1)) MO = MO + 1	STP50300
	IF(MAXO-MSTRT.EQ.1) GO TO 740	STP50400
	IF(MO.EQ.1)AT = AT*AT	STP50500
	IF(WVAR(MO)*AT.GT.WVAR(3)) MO = 3	STP50600
740	MONW = MO	STP50700
	IF(WVAR(MO).GE.1.0045) GO TO 2040	STP50800
	RETURN	STP50900
C		STP51000
C	CHECKS THE PARTITION TO INSURE SMALLEST AVAILABLE STEP	STP51100
C	SIZE IS USED	STP51200
C		STP51300
790	JNW = 1	STP51400
	NEQ(MO)=N	STP51500
800	JA=NPT(INW,JNW)	STP51600
	IF(JNW.EQ.1) GO TO 805	STP51700
	IF(JNW.NE.N) GO TO 810	STP51800
	H=RINT(2)-DPOS(MO,2,JA)	STP51900
	JLD=-1	STP52000
	GO TO 830	STP52100
805	H=DPOS(MO,2,JA)-RINT(1)	STP52200
	JLD=1	STP52300

	GO TO 830	
810	JB=NPT(INW,JNW-1)	STP52400
	JC=NPT(INW,JNW+1)	STP52500
	JLD=1	STP52600
	VA=DPOS(MO,2,JA)-DPOS(MO,2,JB)	STP52700
	IF(VA.LE.0.) GO TO 2050	STP52800
	AB=DPOS(MO,2,JC)-DPOS(MO,2,JA)	STP52900
	IF(AB.LE.0.) GO TO 2050	STP53000
	IF(DABS(VA-AB).LE.DHMIN) GO TO 850	STP53100
	IF(VA.GE.AB) GO TO 820	STP53200
	JLD=-1	STP53300
	H=AB	STP53400
	GO TO 830	STP53500
820	JLD=1	STP53600
	H=VA	STP53700
830	DO 840 ILD=1,N	STP53800
	JB=JNW+JLD*ILD	STP53900
	IF(JB.LE.0.OR.JB.GT.N) GO TO 2060	STP54000
	JB=NPT(INW,JB)	STP54100
	IF(DABS(DABS(DPOS(MO,2,JB)-DPOS(MO,2,JA))-H).LT.HMIN/2.) GO TO 850	STP54200
840	CONTINUE	STP54300
	GO TO 2060	STP54400
850	DPOS(MO,1,JA)=DFLOAT(JLD*(ILD))	STP54500
	ILD=1	STP54600
	JNW=JNW+1	STP54700
	IF(JNW.LE.N) GO TO 800	STP54800
	DO 860 I=1,N	STP54900
860	NPTER(MO,I)=NPT(INW,I)	STP55000
	GO TO 700	STP55100
C		STP55200
C	CHECKS THE OLD PARTITION FOR VALUES. IF NOT PRESENT,	STP55300
C	IT INTERPOLATES	STP55400
C		STP55500
1800	IF(MO.GT.MOLD) GO TO 2070	STP55600
	JC=MAXN+1	STP55700
	N3=1	STP55800
	KLA = 7	STP55900
	GO TO 1930	STP56000
1810	JC=MAXN+2	STP56100
	N3=NUM+2	STP56200
	KLA = 6	STP56300
	GO TO 1930	STP56400
1820	KLA=2	STP56500
	XI=.25	STP56600
	JC=2	STP56700
	GO TO 1850	STP56800
1830	KLA=3	STP56900
	XI=.7500000000	STP57000
	JC = 3	STP57100
	GO TO 1850	STP57200
1840	KLA=1	STP57300
	JC=1	STP57400
	XI=.5	STP57500
1850	N1=1	STP57600
	N2=NUM +2	STP57700
		STP57800

1860	N3=(N1+N2)/2	STP57900
	IF(N3.EQ.N1) GO TO 1890	STP58000
	IF (X(N3) - XI) 1870,1930,1880	STP58100
1870	N1=N3	STP58200
	GO TO 1860	STP58300
1880	N2=N3	STP58400
	GO TO 1860	STP58500
1890	IF(N3 .LT. 1 .OR. N3 .GT. NUM+1) GO TO 2080	STP58600
	ITST = NUM -2	STP58700
	IF(N3 .EQ. 1) ITST = 0	STP58800
	IF(N3 .EQ. NUM + 1) ITST = NUM - 2	STP58900
	DO 1910 I11 = 1,4	STP59000
	I12 = ITST + I11	STP59100
	DO 1900 I13=1,MOLD	STP59200
1900	YT(I13,I11)=Y(1,I13,I12)	STP59300
1910	XT(I11)=X(I12)	STP59400
	CALL DERIV(XT,YT,IDER(1,MO),MO,0,1, 0,MOLD,XI,GAM,HQ(1))	STP59500
	DO 1920 I13 = 1,MO	STP59600
1920	YTEMP(I13,JC)=GAM(I13)	STP59700
	GO TO 1950	STP59800
1930	DO 1940 I13 = 1,MO	STP59900
1940	YTEMP(I13,JC)=Y(1,I13,N3)	STP60000
1950	GO TO (1820,1830,190,210,230,1840,1810),KLA	STP60100
2000	K = -201	STP60200
	RETURN	STP60300
2010	K = -202	STP60400
	RETURN	STP60500
2020	K = -205	STP60600
	RETURN	STP60700
C		STP60800
C	RETURN CODES ARE GENERATED	STP60900
C		STP61000
2030	K = -206	STP61100
	NEQ(MO)=-3	STP61200
	GO TO 700	STP61300
2040	K = -209	STP61400
	RETURN	STP61500
2050	K = -208	STP61600
	RETURN	STP61700
2060	K = -207	STP61800
	RETURN	STP61900
2070	K = -203	STP62000
	RETURN	STP62100
2080	K = -204	STP62200
	RETURN	STP62300
2090	NEQ(MO) = -213	STP62400
	GO TO 700	STP62500
	END	STP62600

SUBROUTINE DERIV(X, FN, IDER, MO, KB, KF, KG, MOLD, XI, GAM, H)

THIS PROGRAM IS USED TO FORM INTERPOLATION POLYNOMIALS FOR LAGRANGIAN, HERMITE, AND TRIMITE INTERPOLATING POLYNOMIALS FOR FUNCTIONAL VALUES AS WELL AS SEVERAL DERIVATIVES. IT IS DIVIDED INTO TWO SECTIONS. ONE SECTION CALCULATES THE INTERPOLATING POLYNOMIALS NEEDED IN ESTIMATING ERRORS. THE OTHER SECTION IS USED TO CALCULATE NEW VALUES FOR FUNCTIONS AND ASSOCIATED DERIVATIVES WHEN NEW PARTITION POINTS ARE DESIRED

THE VARIABLES IN THIS SUBPROGRAM ARE

X	CONTAINS INTERPOLATING POINTS	
Y	CONTAINS FUNCTION AND DERIVATIVE VALUES	
IDER	TELLS WHICH DERIVATIVES ARE REQUIRED	
MO	ORDER OF METHOD (POSSIBLY NEW)	
MOLD	OLD MO	
ALPHA	CONTAINS THE FACTORS $X-A(I)$ AND $A(J)-A(I)$	
	FOR EACH POINT AND COEFFICIENT	
DIV	CONTAINS VARIOUS FUNCTIONAL VALUES FOR EACH POINT	
	(1,I) $L'''(X)$	
	(2,I) $L'(X)$	
	(3,I) $L'(I)(X(I))$	
	(4,I) $L''(X)$	
	(5,I) $L''(I)(X(I))$	
	(6,I) DENOMINATOR OF $L(X)$	
	(7,I) A'	THOSE FROM
	(8,I) $(A \text{ BAR})'$	7 TO 15
	(9,I) $(A \text{ DOUBLE BAR})'$	ARE ALSO
	(10,I) A''	USED TO
	(11,I) $(A \text{ BAR})''$	STORE
	(12,I) $(A \text{ DOUBLE BAR})''$	TEMPORARY
	(13,I) A	RESULTS IN
	(14,I) $A \text{ BAR}$	INTERPOLATION
	(15,I) $A \text{ DOUBLE BAR}$	
	(16,I) $L(X)$	
COEF	CONTAINS THE COEFFICIENTS OF THE RESULTING	
KB	1 IF COEFFICIENTS DESIRED, 0 OTHERWISE	
	INTERPOLATING POLYNOMIALS	
KF	1-INTERPOLATE 0-ERROR ESTIMATE	
KG	1-AT PARTITION POINT 0-NOT AT	
XI	EVALUATION POINT	
GAM	RESULTS OF INTERPOLATION	
DVY	RESULTS OF ERROR ESTIMATES	
H	DISTANCE USED IN ANALOG. HSQ, HSQQ, & HSQC ARE	
	$-(H**2), H**4, H**6$ RESPECTIVELY	

THE REST OF THE VARIABLES ARE POINTERS AND COUNTERS

IMPLICIT REAL * 8 (A-H, Q-Z)
 DIMENSION X(1), FN(3,5), IDER(4), GAM(3)
 COMMON /DVR/ DVY(4), COEF(4,3,3), ALPHA(2,4,3), DIV(16,5)
 IF(KF.EQ.1) GO TO 260
 HSQ = - H * H

DRV10000
 DRV10100
 DRV10200
 DRV10300
 DRV10400
 DRV10500
 DRV10600
 DRV10700
 DRV10800
 DRV10900
 DRV11000
 DRV11100
 DRV11200
 DRV11300
 DRV11400
 DRV11500
 DRV11600
 DRV11700
 DRV11800
 DRV11900
 DRV12000
 DRV12100
 DRV12200
 DRV12300
 DRV12400
 DRV12500
 DRV12600
 DRV12700
 DRV12800
 DRV12900
 DRV13000
 DRV13100
 DRV13200
 DRV13300
 DRV13400
 DRV13500
 DRV13600
 DRV13700
 DRV13800
 DRV13900
 DRV14000
 DRV14100
 DRV14200
 DRV14300
 DRV14400
 DRV14500
 DRV14600
 DRV14700
 DRV14800
 DRV14900
 DRV15000
 DRV15100
 DRV15200

```

IF(MO .LT. 2 ) GO TO 5
HSQQ=HSQ*HSQ
IF(MO .LT. 3 ) GO TO 5
HSQC=HSQQ*HSQ
C
C TEST IF DERIVATIVES ARE USED
C
5 KA = 0
DO 10 IA=1,4
DVI(IA)=0.
10 KA=KA+IDER(IA)
IF(KA.EQ.0 ) RETURN
C
C SETS FOR HIGHEST LAGRANGIAN DERIVATIVE
C
IF(MO.NE.1.OR.IDER(2).NE.1) GO TO 20
N1=5
K=2
VA=12.
V=24.
GO TO 30
C
C SETS FOR THE LOWER LAGRANGIAN DERIVATIVE
C
20 IF(MO.EQ.1.AND.IDER(1).EQ. 0) RETURN
K=1
N1=4
VA=6.
V=6.
C
C OBTAINS DENOMINATOR OF LAGRANGIAN POLYNOMIALS
C
30 DO 40 IA=1,N1
DIV(6,IA)=1.
DO 40 IB=1,N1
IF(IA.EQ. IB) GO TO 40
DIV(6,IA)=DIV(6,IA)*(X(IA)-X(IB))
40 CONTINUE
IF(KF.NE.1) GO TO 45
KSAVE=KA
GO TO 70
C
C THE FOLLOWING FORMS L'''(X)
C
45 DO 50 IA=1,N1
50 DIV(1,IA)=V/DIV(6,IA)
IF(MO.NE.1) GO TO 70
C
C CALCULATES ERROR ESTIMATES FOR LAGRANGIAN
C
DVI(K)=0.
KA = 4*(K-1)
DO 60 IA=1,N1
IF(KB .EQ. 0 ) GO TO 58
IB = KA + IA

```

DRV15300
 DRV15400
 DRV15500
 DRV15600
 DRV15700
 DRV15800
 DRV15900
 DRV16000
 DRV16100
 DRV16200
 DRV16300
 DRV16400
 DRV16500
 DRV16600
 DRV16700
 DRV16800
 DRV16900
 DRV17000
 DRV17100
 DRV17200
 DRV17300
 DRV17400
 DRV17500
 DRV17600
 DRV17700
 DRV17800
 DRV17900
 DRV18000
 DRV18100
 DRV18200
 DRV18300
 DRV18400
 DRV18500
 DRV18600
 DRV18700
 DRV18800
 DRV18900
 DRV19000
 DRV19100
 DRV19200
 DRV19300
 DRV19400
 DRV19500
 DRV19600
 DRV19700
 DRV19800
 DRV19900
 DRV20000
 DRV20100
 DRV20200
 DRV20300
 DRV20400
 DRV20500
 DRV20600
 DRV20700

	COEF(IB,1,1)=DIV(1,IA)/VA *HSQ	DRV20800
	GO TO 60	DRV20900
58	DVY(K) = DVY(K) + DIV(1,IA)*FN(1,IA)	DRV21000
60	CONTINUE	DRV21100
	IF(KB.EQ.1) GO TO 62	DRV21200
	DVY(K)=DVY(K)/VA *HSQ	DRV21300
62	IF(K.EQ. 1) RETURN	DRV21400
	GO TO 20	DRV21500
C		DRV21600
C	CALCULATES THE FACTORS USED IN THE NUMERATOR	DRV21700
C		DRV21800
70	DO 80 IA=1,N1	DRV21900
	KA=1	DRV22000
	DO 80 IB=1,N1	DRV22100
	IF(IA.EQ. IB) GO TO 80	DRV22200
	ALPHA(1,IA,KA)=XI-X(IB)	DRV22300
	ALPHA(2,IA,KA)=X(IA)-X(IB)	DRV22400
	KA=KA+1	DRV22500
80	CONTINUE	DRV22600
	IF(KF.EQ.1) KA=KSAVE	DRV22700
	IF(KF.EQ.1) GO TO (270,90,120),MOLD	DRV22800
	KA=2	DRV22900
C		DRV23000
C	CALCULATES L'(X) OR L'(I)(X(I))	DRV23100
C		DRV23200
90	DO 100 IA=1,N1	DRV23300
100	DIV(KA+1,IA)=(ALPHA(KA,IA,1)*(ALPHA(KA,IA,2)+ALPHA(KA,IA,3))+ALPHA(	DRV23400
	1(KA,IA,2)*ALPHA(KA,IA,3))/DIV(6,IA)	DRV23500
	IF(KF.EQ.1) GO TO 280	DRV23600
	IF(KA.EQ.1) GO TO 220	DRV23700
	IF(MO.LE.2) GO TO 140	DRV23800
C		DRV23900
C	CALCULATES L''(X) OR L''(I)(X(I))	DRV24000
C		DRV24100
110	KA=1	DRV24200
120	DO 130 IA=1,N1	DRV24300
130	DIV(KA+3,IA)=(2.*(ALPHA(KA,IA,1)+ALPHA(KA,IA,2)+ALPHA(KA,IA,3)))/D	DRV24400
	1IV(6,IA)	DRV24500
	IF(KF.EQ.1) GO TO 90	DRV24600
	IF(KA.EQ.2) GO TO 200	DRV24700
	IF(MO.EQ.2) GO TO 180	DRV24800
	KA=2	DRV24900
	GO TO 120	DRV25000
C		DRV25100
C	CALCULATES A' FOR HERMITE	DRV25200
C		DRV25300
140	DO 150 IA=1,N1	DRV25400
	DIV(7,IA)=-2.*DIV(3,IA)	DRV25500
150	DIV(8,IA)=1.	DRV25600
	IF(IDER(2).NE.0) GO TO 110	DRV25700
C		DRV25800
C	CALCULATES THIRD DERIVATIVE FOR HERMITE	DRV25900
C		DRV26000
160	IF(IDER(3).EQ.0) RETURN	DRV26100
	DO 170 IA=1,N1	DRV26200

```

V=140.00000*DIV(1,IA)*DIV(1,IA)
COEF(IA,2,1)=DIV(7,IA)*V
COEF(IA,2,2)=DIV(8,IA)*V
IF(KB.EQ.0) GO TO 171
COEF(IA,2,1) = COEF(IA,2,1)/840.000000 * HSQQ
COEF(IA,2,2) = COEF(IA,2,2)/840.000000 * HSQQ
GO TO 170
171 DVY(3)=DVY(3)+FN(1,IA)*COEF(IA,2,1)+FN(2,IA)*COEF(IA,2,2)
170 CONTINUE
IF(KB.EQ.1) RETURN
DVY(3) = DVY(3) / 840.000000 * HSQQ
RETURN

C
C CALCULATE A FOR HERMITE
C
180 DO 190 IA=1,N1
DIV(13,IA)=1-2.*(XI-X(IA)) *DIV(3,IA)
DIV(14,IA)=(XI-X(IA))
V=20.000000*DIV(1,IA)*DIV(1,IA)
VA=120.000000*DIV(4,IA)*DIV(1,IA)

C
C CALCULATES SECOND DERIVATIVE FOR HERMITE
C
COEF(IA,1,1)=V*DIV(13,IA)+VA*DIV(7,IA)
COEF(IA,1,2)=V*DIV(14,IA)+VA*DIV(8,IA)
IF(KB.EQ.0) GO TO 191
COEF(IA,1,1)=COEF(IA,1,1)/360.000000 *HSQQ
COEF(IA,1,2)=COEF(IA,1,2)/360.000000 *HSQQ
GO TO 190
191 DVY(2)=DVY(2)+FN(1,IA)*COEF(IA,1,1)+FN(2,IA)*COEF(IA,1,2)
190 CONTINUE
IF ( KB .EQ.1) GO TO 160
DVY(2)=DVY(2)/360.000000 *HSQQ
GO TO 160

C
C CALCULATES A',A'' FOR TRIMITE
C
200 DO 210 IA=1,N1
DIV(11,IA)=-6.0000000*DIV(3,IA)
DIV(10,IA)=-2.*DIV(11,IA)*DIV(3,IA)-3.00000000*DIV(5,IA)
DIV(12,IA)=1.
DIV(7,IA)=DIV(10,IA)*(XI-X(IA)) -3.00000000*DIV(3,IA)
DIV(8,IA)=1.+(XI-X(IA)) *DIV(11,IA)
210 DIV(9,IA)=XI-X(IA)
IF(KF.EQ.1) GO TO 220
IF(IDER(3).EQ.0) GO TO 240
KA=1
GO TO 90

C
C CALCULATES A FOR TRIMITE
C
220 DO 230 IA=1,N1
DIV(15,IA)=.5*(XI-X(IA))*(XI-X(IA))
DIV(13,IA)=1.+DIV(15,IA)*DIV(10,IA)-3.0000000*(XI-X(IA))
1,IA)

```

DRV2630
 DRV2640
 DRV2650
 DRV2660
 DRV2670
 DRV2680
 DRV2690
 DRV2700
 DRV2710
 DRV2720
 DRV2730
 DRV2740
 DRV2750
 DRV2760
 DRV2770
 DRV2780
 DRV2790
 DRV2800
 DRV2810
 DRV2820
 DRV2830
 DRV2840
 DRV2850
 DRV2860
 DRV2870
 DRV2880
 DRV2890
 DRV2900
 DRV2910
 DRV2920
 DRV2930
 DRV2940
 DRV2950
 DRV2960
 DRV2970
 DRV2980
 DRV2990
 DRV3000
 DRV3010
 DRV3020
 DRV3030
 DRV3040
 DRV3050
 DRV3060
 DRV3070
 DRV3080
 DRV3090
 DRV3100
 DRV3110
 DRV3120
 DRV3130
 DRV3140
 DRV3150
 *DIV(3)DRV31600
 DRV31700

	DIV(14,IA)=(XI -X(IA)) +DIV(15,IA)*DIV(11,IA)	DRV31800
	IF(KF.EQ.1) GO TO 230	DRV31900
	DIV(6,1)=DIV(1,IA)*DIV(1,IA)	DRV32000
C		DRV32100
C	CALCULATES THIRD DERIVATIVE FOR TRIMITE	DRV32200
C		DRV32300
	V=1680.0000*DIV(1,IA)*DIV(6,1)	DRV32400
	VA=15120.000*DIV(4,IA)*DIV(6,1)	DRV32500
	VB=22680.000*DIV(4,IA)*DIV(1,IA)*DIV(4,IA)+15120.000*DIV(2,IA)*DIV(6,1)	DRV32600
	COEF(IA,1,1)=V*DIV(13,IA)+VA*DIV(7,IA)+DIV(10,IA)*VB	DRV32700
	COEF(IA,1,2)=V*DIV(14,IA)+VA*DIV(8,IA)+DIV(11,IA)*VB	DRV32800
	COEF(IA,1,3)=V*DIV(15,IA)+VA*DIV(9,IA)+DIV(12,IA)*VB	DRV32900
	IF(KB.EQ.0) GO TO 229	DRV33000
	DO 225 KFL =1,3	DRV33100
225	COEF(IA,1,KFL)=COEF(IA,1,KFL)/60480.000 *HSQC	DRV33200
	GO TO 230	DRV33300
229	DVY(3)=DVY(3)+FN(1,IA)*COEF(IA,1,1)+FN(2,IA)*COEF(IA,1,2)+FN(3,IA)*COEF(IA,1,3)	DRV33400
230	CONTINUE	DRV33500
	IF(KF.EQ.1) GO TO 450	DRV33600
	IF (KB .EQ.1) GO TO 240	DRV33700
	DVY(3)=DVY(3)/60480.000 *HSQC	DRV33800
C		DRV33900
C	CALCULATES FOURTH DERIVATIVE FOR TRIMITE	DRV34000
C		DRV34100
240	IF(IDER(4).EQ.0) RETURN	DRV34200
	DO 251 IA=1,N1	DRV34300
	V=DIV(1,IA)*DIV(1,IA)	DRV34400
	VA=16800.000*V*DIV(1,IA)	DRV34500
	VB=75600.000*V*DIV(4,IA)	DRV34600
	COEF(IA,2,1)=VA*DIV(7,IA)+VB*DIV(10,IA)	DRV34700
	COEF(IA,2,2)=VA*DIV(8,IA)+VB*DIV(11,IA)	DRV34800
	COEF(IA,2,3)=VA*DIV(9,IA)+VB*DIV(12,IA)	DRV34900
	IF(KB.EQ.0) GO TO 250	DRV35000
	DO 245 KFL=1,3	DRV35100
245	COEF(IA,2,KFL)=COEF(IA,2,KFL)/151200.00 *HSQC	DRV35200
	GO TO 251	DRV35300
250	DVY(4)=DVY(4)+FN(1,IA)*COEF(IA,2,1)+FN(2,IA)*COEF(IA,2,2)+FN(3,IA)*COEF(IA,2,3)	DRV35400
251	CONTINUE	DRV35500
	IF (KB .EQ.1) RETURN	DRV35600
	DVY(4)=DVY(4)/151200.00 *HSQC	DRV35700
	RETURN	DRV35800
260	IF(MOLD .GE. MO .AND. KG .EQ.1) GO TO 520	DRV35900
	KA=2	DRV36000
	N1=4	DRV36100
	GO TO 30	DRV36200
270	KA=1	DRV36300
	GO TO (290,90,120),MO	DRV36400
C		DRV36500
C		DRV36600
C	CALCULATES L(X)	DRV36700
		DRV36800
280	IF(KA.EQ.2) GO TO 270	DRV36900
290	DO 300 IA=1,N1	DRV37000
		DRV37100
		DRV37200

```

300  DIV(16,IA)=ALPHA(1,IA,1)*ALPHA(1,IA,2)*ALPHA(1,IA,3)/DIV(6,IA)  DRV37300
    GO TO (310,380,200),MOLD  DRV37400
310  GO TO (360,340,320),MO  DRV37500
C  DRV37600
C  INTERPOLATES THE SECOND DERIVATIVE USING LAGRANGIAN  DRV37700
C  DRV37800
320  GAM(3)=0.  DRV37900
    DO 330  IA=1,N1  DRV38000
    COEF(IA,1,3)=DIV(4,IA)  DRV38100
    IF(KB .EQ. 1) GO TO 330  DRV38200
    GAM(3)=GAM(3)+DIV(4,IA)*FN(1,IA)  DRV38300
330  CONTINUE  DRV38400
C  DRV38500
C  INTERPOLATES THE FIRST DERIVATIVE USING LAGRANGIAN  DRV38600
C  DRV38700
340  GAM(2)=0.  DRV38800
    DO 350  IA=1,N1  DRV38900
    COEF(IA,1,2)=DIV(2,IA)  DRV39000
    IF(KB .EQ. 1) GO TO 350  DRV39100
    GAM(2)=      GAM(2)      +DIV(2,IA)*FN(1,IA)  DRV39200
350  CONTINUE  DRV39300
C  DRV39400
C  INTERPOLATES THE FUNCTIONAL VALUE USING LAGRANGIAN  DRV39500
C  DRV39600
360  GAM(1) = 0.  DRV39700
    DO 370  IA=1,N1  DRV39800
    COEF(IA,1,1)=DIV(16,IA)  DRV39900
    IF( KB .EQ. 1) GO TO 370  DRV40000
    GAM(1) = GAM(1) + DIV(16,IA) * FN(1,IA)  DRV40100
370  CONTINUE  DRV40200
    RETURN  DRV40300
C  DRV40400
C  DEVELOPS COEFFICIENTS  DRV40500
C  DRV40600
380  DO 390  IA=1,N1  DRV40700
    DIV(7 ,IA)=-2.*DIV(3,IA)  DRV40800
    DIV(13,IA)=1.+(XI-X(IA))      *DIV( 7,IA)  DRV40900
    DIV(8,IA)=1.  DRV41000
390  DIV(14,IA)=XI-X(IA)  DRV41100
    GO TO (430,420,400),MO  DRV41200
C  DRV41300
C  INTERPOLATES SECOND DERIVATIVE USING HERMITE  DRV41400
C  DRV41500
400  GAM(3)=0.  DRV41600
    DO 410  IA=1,N1  DRV41700
    V=2.*(DIV(2,IA)*DIV(2,IA)+DIV(16,IA)*DIV(4,IA))  DRV41800
    VA=4.*DIV(16,IA)*DIV(2,IA)  DRV41900
    COEF(IA,1,3)=V*DIV(13,IA)+VA*DIV(7,IA)  DRV42000
    COEF(IA,2,3)=V*DIV(14,IA)+VA  DRV42100
    IF(KG .EQ. 1) GO TO 410  DRV42200
    GAM(3)=GAM(3)      +COEF(IA,1,3)*FN(1,IA)+COEF(IA,2,3)*FN(2,  DRV42300
1IA)  DRV42400
410  CONTINUE  DRV42500
C  DRV42600
C  INTERPOLATES FIRST  DERIVATIVE USING HERMITE  DRV42700

```

C	420	GAM(2)=0.	DRV42800
		DO 425 IA=1,N1	DRV42900
		V=2.*DIV(16,IA)*DIV(2,IA)	DRV43000
		VA=DIV(16,IA)*DIV(16,IA)	DRV43100
		COEF(IA,1,2)=V*DIV(13,IA)+VA*DIV(7,IA)	DRV43200
		COEF(IA,2,2)=V*DIV(14,IA)+VA	DRV43300
		IF(KB .EQ. 1) GO TO 425	DRV43400
		GAM(2) = GAM(2) + COEF(IA,1,2) *FN(1,IA) + COEF(IA,2,2)*FN(2,IA)	DRV43500
	425	CONTINUE	DRV43600
C			DRV43700
C		INTERPOLATES THE FUNCTIONAL VALUE USING HERMITE	DRV43800
C			DRV43900
	430	GAM(1)=0.	DRV44000
		DO 440 IA=1,N1	DRV44100
		VA=DIV(16,IA)*DIV(16,IA)	DRV44200
		COEF(IA,1,1)=VA*DIV(13,IA)	DRV44300
		COEF(IA,2,1)=VA *DIV(14,IA)	DRV44400
		IF(KB .EQ. 1) GO TO 440	DRV44500
		GAM(1) = GAM(1) + COEF(IA,1,1)*FN(1,IA) + COEF(IA,2,1)*FN(2,IA)	DRV44600
	440	CONTINUE	DRV44700
		RETURN	DRV44800
	450	GO TO (500,480,460),MO	DRV44900
C			DRV45000
C		INTERPOLATES SECOND DERIVATIVE USING TRIMITE	DRV45100
C			DRV45200
	460	GAM(3)=0.	DRV45300
		DO 470 IA=1,N1	DRV45400
		VB=DIV(16,IA)*DIV(16,IA)	DRV45500
		VA=6.0000000*VB*DIV(2,IA)	DRV45600
		V=3.0000000*VB*DIV(4,IA)+6.0000000*DIV(16,IA)*DIV(2,IA)*DIV(2,IA)	DRV45700
		VB=VB*DIV(16,IA)	DRV45800
		COEF(IA,1,3)=V*DIV(13,IA)+VA*DIV(7,IA)+VB*DIV(10,IA)	DRV45900
		COEF(IA,2,3)=V*DIV(14,IA)+VA*DIV(8,IA)+VB*DIV(11,IA)	DRV46000
		COEF(IA,3,3)=V*DIV(15,IA)+VA*DIV(9,IA)+VB*DIV(12,IA)	DRV46100
		IF(KB .EQ. 1) GO TO 470	DRV46200
		GAM(3) = GAM(3) + COEF(IA,1,3)*FN(1,IA) + COEF(IA,2,3)*FN(2,IA)	DRV46300
		1 + COEF(IA,3,3)*FN(3,IA)	DRV46400
	470	CONTINUE	DRV46500
C			DRV46600
C		INTERPOLATES FIRST DERIVATIVE USING TRIMITE	DRV46700
C			DRV46800
	480	GAM(2)=0.	DRV46900
		DO 490 IA=1,N1	DRV47000
		VB=DIV(16,IA)*DIV(16,IA)	DRV47100
		V=3.0000000*VB*DIV(2,IA)	DRV47200
		VA=DIV(16,IA)*VB	DRV47300
		COEF(IA,1,2)=V*DIV(13,IA)+VA*DIV(7,IA)	DRV47400
		COEF(IA,2,2)=V*DIV(14,IA)+VA*DIV(8,IA)	DRV47500
		COEF(IA,3,2)=V*DIV(15,IA)+VA*DIV(9,IA)	DRV47600
		IF(KB .EQ. 1) GO TO 490	DRV47700
		GAM(2) = GAM(2) + COEF(IA,1,2)*FN(1,IA) + COEF(IA,2,2)*FN(2,IA)	DRV47800
		1 + COEF(IA,3,2)*FN(3,IA)	DRV47900
	490	CONTINUE	DRV48000
C			DRV48100
			DRV48200

C INTERPOLATES THE FUNCTIONAL VALUE USING TRIMITE

```

C
500 GAM(1)=0.
    DO 510 IA=1,N1
      V=DIV(16,IA)*DIV(16,IA)*DIV(16,IA)
      COEF(IA,1,1)=V*DIV(13,IA)
      COEF(IA,2,1)=V*DIV(14,IA)
      COEF(IA,3,1)=V*DIV(15,IA)
      IF(KB .EQ. 1) GO TO 510
      GAM(1) = GAM(1) + COEF(IA,1,1)*FN(1,IA) + COEF(IA,2,1)*FN(2,IA)
      1 + COEF(IA,3,1)*FN(3,IA)
510 CONTINUE
    RETURN
520 WRITE(6,1CC01)
10001 FORMAT(' INVALID ORDERS')
    RETURN
    END

```

DRV48300
 DRV48400
 DRV48500
 DRV48600
 DRV48700
 DRV48800
 DRV48900
 DRV49000
 DRV49100
 DRV49200
 DRV49300
 DRV49400
 DRV49500
 DRV49600
 DRV49700
 DRV49800
 DRV49900

SUBROUTINE DECOMP(N,UL,J5,K7)
 IMPLICIT REAL *8(A-H,O-Z)

THIS SUBPROGRAM DECOMPOSES THE MATRIX INTO UPPER AND LOWER
 TRIANGULAR MATRICES. THESE HOLD THE OPERATIONS PERFORMED IN
 THE FIRST HALF OF A GAUSSIAN ELIMINATION PROCEDURE.

THE VARIABLES IN THIS SUBPROGRAM ARE

UL THE ARRAY
 N THE NUMBER OF VARIABLES
 J5 A RETURN CODE 1 IF SUCCESSFUL, -1 IF NOT
 K7 THE NUMBER OF DIAGONALS IN THE MATRIX
 K3 POINTS TO POSITION OF THE MAIN DIAGONAL
 K2 NUMBER OF SUBDIAGONALS

THE REST OF THE VARIABLES ACT AS COUNTERS OR AS POINTERS
 TO POSITIONS IN UL

```

DIMENSION UL(K7,1)
COMMON /CNT/ ISTEP,NUMDE,NUMSO
NUMDE=NUMDE+1
J5=1
K2=K7/2
K3=K2+1
DO 10 I=K3,N
  K = I + 1
  I1=I-K2
  DO 10 J=1,K2
    K = K - 1
    IF(DABS(UL(K3,I1)).LT.1.D-20) GO TO 15

```

CALCULATES THE FACTOR USED TO SUBTRACT ONE EQUATION FROM
 ANOTHER

DCP10000
 DCP10100
 DCP10200
 DCP10300
 DCP10400
 DCP10500
 DCP10600
 DCP10700
 DCP10800
 DCP10900
 DCP11000
 DCP11100
 DCP11200
 DCP11300
 DCP11400
 DCP11500
 DCP11600
 DCP11700
 DCP11800
 DCP11900
 DCP12000
 DCP12100
 DCP12200
 DCP12300
 DCP12400
 DCP12500
 DCP12600
 DCP12700
 DCP12800
 DCP12900
 DCP13000
 DCP13100
 DCP13200

	UL(J,K)=UL(J,K)/UL(K3,I1)		DCP13300
	J1=J+1		DCP13400
	J2=J+ K2		DCP13500
	L1 = K3		DCP13600
	DO 10 L = J1,J2		DCP13700
	L1 = L1 + 1		DCP13800
C			DCP13900
C	SUBTRACTS CORRESPONDING ELEMENTS FROM THE TWO EQUATIONS		DCP14000
C			DCP14100
10	UL(L,K)=UL(L,K)-UL(L1,I1)*UL(J,K)		DCP14200
	IF(K2.EQ.1) RETURN		DCP14300
C			DCP14400
C	THE FOLLOWING REPEATS THE CODE ABOVE TO HANDLE THE SITUATION		DCP14500
C	NEAR THE END OF THE MATRIX.		DCP14600
			DCP14700
	DO 20 I=2,K2		DCP14800
	I1 = N + I		DCP14900
	K5 = I1 - K3		DCP15000
	I2 = K2 - I		DCP15100
	DO 20 J=I,K2		DCP15200
	K = I1 - J		DCP15300
	IF(DABS(UL(K3,K5)).LT.1.0-20) GO TO 15		DCP15400
	UL(J,K)=UL(J,K)/UL(K3,K5)		DCP15500
	J1=J+1		DCP15600
	J2 = I2 + J1		DCP15700
	L1 = K3		DCP15800
	DO 20 L=J1,J2		DCP15900
	L1 = L1 + 1		DCP16000
20	UL(L,K)=UL(L,K)-UL(L1,K5)*UL(J,K)		DCP16100
	RETURN		DCP16200
15	J5=-1		DCP16300
	RETURN		DCP16400
	END		DCP16500

SUBROUTINE SOLVE(N,UL,B,X,K7,ID,K)
 IMPLICIT REAL *8(A-H,O-Z)

THIS PROGRAM USES THE DECOMPOSED MATRIX FROM DECOMP TO
 PERFORM THE GAUSSIAN ELIMINATION

THE VARIABLES IN THIS SUBPROGRAM ARE

UL	THE ARRAY	SLV10000
N	THE NUMBER OF VARIABLES	SLV10100
K7	THE NUMBER OF DIAGONALS IN THE MATRIX	SLV10200
K3	POINTS TO POSITION OF THE MAIN DIAGONAL	SLV10300
K2	NUMBER OF SUBDIAGONALS	SLV10400
B	THE CONSTANT VECTOR OF THE EQUATION TO BE SOLVED	SLV10500
X	THE RESULT VECTOR	SLV10600
ID	THE NUMBER OF ROWS IN THE RESULT VECTOR	SLV10700
K	THE ROW IN RESULT VECTOR WHERE ANSWER IS PLACED	SLV10800
THE REST OF THE VARIABLES ACT AS COUNTERS OR AS POINTERS		SLV10900
		SLV11000
		SLV11100
		SLV11200
		SLV11300
		SLV11400
		SLV11500
		SLV11600

C	TO POSITIONS IN UL	SLV11700
C		SLV11800
	DIMENSION UL(K7,1),X(ID,1),B(1)	SLV11900
	COMMON /CNT/ ISTEP,NUMDE,NUMSO	SLV12000
	NN = N + 1	SLV12100
	NUMSO=NUMSO+1	SLV12200
C		SLV12300
C	FIRST THE RESULT VECTOR IS INITIALIZED TO THE CONSTANT VECTOR	SLV12400
C		SLV12500
	DO1 I=1,N	SLV12600
1	X(K,I)=B(I)	SLV12700
C		SLV12800
C		SLV12900
	K2=K7/2	SLV13000
	K3=K2+1	SLV13100
	K4=K2-1	SLV13200
	IF(K2.EQ.1) GOTO9	SLV13300
C		SLV13400
C	THIS SECTION PERFORMS THE OPERATIONS OF THE FIRST PART OF THE	SLV13500
C	GAUSSIAN ELIMINATION ON THE CONSTANT VECTOR	SLV13600
C		SLV13700
	DO2 I =2,K2	SLV13800
	I1= I-1	SLV13900
	J3 = K3 - I	SLV14000
	DO 2 J=1,I1	SLV14100
	J2 = J3 + J	SLV14200
2	X(K,I)=X(K,I)-X(K,J)*UL(J2 ,I)	SLV14300
9	DO 3 I=K3,N	SLV14400
	DO3 J =1,K2	SLV14500
	J2=I-J	SLV14600
	J3=K3-J	SLV14700
3	X(K,I)=X(K,I)-X(K,J2)*UL(J3,I)	SLV14800
	IF(K2.GT.1) GO TO 4	SLV14900
	X(K,N)=X(K,N)/UL(2,N)	SLV15000
	GO TO 5	SLV15100
4	X(K,N)=X(K,N)/UL(K3,N)	SLV15200
C		SLV15300
C	THIS SECTION HANDLES THE BACK SUBSTITUTION	SLV15400
C		SLV15500
	DO 6 J =1,K4	SLV15600
	NJ=N-J	SLV15700
	DO 7 I=1,J	SLV15800
	KJ=K3+I	SLV15900
	NJ1=NJ+I	SLV16000
7	X(K,NJ)=X(K,NJ)-UL(KJ,NJ)*X(K,NJ1)	SLV16100
6	X(K,NJ) =X(K,NJ)/UL(K3,NJ)	SLV16200
5	DO 10 IBK=K3,N	SLV16300
	I = NN - IBK	SLV16400
	DO 8 J = 1,K2	SLV16500
	JK=K3+J	SLV16600
	IJ= I+J	SLV16700
8	X(K,I)=X(K,I)-UL(JK,I)*X(K,IJ)	SLV16800
10	X(K,I)=X(K,I)/UL(K3,I)	SLV16900
	RETURN	SLV17000
	END	SLV17100

```

SUBROUTINE DIFFUN(X,Y,T,DIVER,MO,INT,N,IDER,ERR,DY,L,L2,L3)
THE PURPOSE OF THIS PROGRAM IS TO PROVIDE
  1. THE TIME DERIVATIVE AS THE RESULT OF THE
      DIFFERENTIAL EQUATION DURING INTEGRATION
  2. THE TIME DERIVATIVE WHEN STARTING
  3. THE TIME DERIVATIVE FOR THE CALCULATION OF THE
      JACOBIAN
IN THE FIRST CASE L = L2 = 0
IN THE SECOND CASE L=1,L2=0 AND THE PROGRAM ESTIMATES THE TIME
DERIVATIVE OF THE BOUNDARY POINTS BY EXTRAPOLATION FROM INTERIOR
POINTS.
IN THE THIRD CASE L=0,L2=1 AND THE ERROR ESTIMATION CALCULATIONS
ARE BYPASSED.
THE VARIABLES IN THIS SUBPROGRAM ARE
  X      HOLDS THE PARTITION POINTS
  Y      HOLDS THE FUNCTIONAL VALUES
  T      TIME
  DIVER  CONTAINS THE COEFFICIENTS OF ERROR ESTIMATE
          INTERPOLATING POLYNOMIALS
  MO     ORDER OF THE METHOD
  INT    PROVIDE POINTS OF ANALOG (INT(1,I)) AND THE
          STARTING POINT OF ERROR ESTIMATE ANALOG (INT(2,I))
  N      NUMBER OF EQUATIONS
  IDER   CONTAINS NAMES OF DERIVATIVES FOR ALL ORDERS
  ERR    CONTAINS ERROR ESTIMATE VECTOR
  DY     CONTAINS THE TIME DERIVATIVE
  L      INDICATES IF FINDING TIME DERIVATIVE AT BOUNDARY
  L2     INDICATES JACOBIAN CALCULATION
  DV     TEMPORARILY STORES SPACE DERIVATIVES
  DVP    TEMPORARILY STORES ERROR ESTIMATE FOR DV
THE REST OF THE VARIABLES ACT AS COUNTERS OR POINTERS.

IMPLICIT REAL * 8 (A-H,Q-Z)
DIMENSION SV(6)
DIMENSION X(1),Y(5,MO,1),DY(MO,1),DIVER(9,MO,1),INT(2,1),IDER(4),
1DV(4),DVP(4),ER(4),DJ(4),ERR(MO,1)
DIMENSION XT(5),XS(5),YT(3,5),YS(3,5)
COMMON /DVR/ DVP(4),COEF(4,3,3),ALPHA(2,4,3),DIV(16,5)

THIS SECTION DETERMINES THE TIME DERIVATIVES AT THE BOUNDARIES

IF(L3.EQ.0) GO TO 4
DO 3 IA=1,MO
SV(IA)=Y(1,IA,1)
3 SV(IA+3)=Y(1,IA,N)
4 CONTINUE
CALL BDRY(X,Y,T,MO,N,DY)
DO 5 IA = 1,MO

```

DFN10000
 DFN10100
 DFN10200
 DFN10300
 DFN10400
 DFN10500
 DFN10600
 DFN10700
 DFN10800
 DFN10900
 DFN11000
 DFN11100
 DFN11200
 DFN11300
 DFN11400
 DFN11500
 DFN11600
 DFN11700
 DFN11800
 DFN11900
 DFN12000
 DFN12100
 DFN12200
 DFN12300
 DFN12400
 DFN12500
 DFN12600
 DFN12700
 DFN12800
 DFN12900
 DFN13000
 DFN13100
 DFN13200
 DFN13300
 DFN13400
 DFN13500
 DFN13600
 DFN13700
 DFN13800
 DFN13900
 DFN14000
 DFN14100
 DFN14200
 DFN14300
 DFN14400
 DFN14500
 DFN14600
 DFN14700
 DFN14800
 DFN14900

```

ERR(IA,1) = DY(IA,1)
ERR(IA,N) = DY(IA,N)
5 CONTINUE
C
C THIS SECTION DETERMINES WHICH POINTS ARE USED IN THE DERIVATIVE
C
NM1=N-1
DO 100 IA = 2,NM1
IG=IA
IF(INT(1,IA).LT.0) GO TO 10
IB=IA+INT(1,IA)
IC=IA-1
C
C DETERMINES STARTING POINT FOR ERR AND THE POWERS OF SPACING
C
GO TO 20
10 IB=IA+1
IC=IA+INT(1,IA)
20 ID=INT(2,IA)
XI=X(IA)-X(IC)
IF(XI .EQ. XITMP .AND. MO .EQ. MOLD) GO TO 25
MOLD = MO
XITMP = XI
XIS=XI*XI
IF (MO .LT. 2) GO TO 25
XIC=XI*XIS
IF (MO .LT. 3) GO TO 25
XIF=XI*XIC
25 GO TO (30,70,80),MO
C
C SETS PARAMETERS AND FORMS DERIVATIVES FOR LAGRANGIAN
C
30 IF(IDER(1).EQ.1) DV(1)=(Y(1,1,IB)-Y(1,1,IC))/(2.*XI)
IF(IDER(2).EQ.1) DV(2)=(Y(1,1,IB)-2.*Y(1,1,IA)+Y(1,1,IC))/XIS
DVP(1)=0.
DVP(2)=0.
C
C USES COEFFICIENTS OF INTERPOLATING POLYNOMIAL AND CALCULATES
C ERROR DERIVATIVES.
C
40 IF (L2 .EQ. 1) GO TO 65
DO 50 IE=1,MO
DO 50 IF=1,4
IH=ID+IF-1
IF(IDER(MO).EQ.1) DVP(MO)=DIVER(IF,IE,IA)*Y(1,IE,IH)+DVP(MO)
IF(IDER(MO+1).EQ.1) DVP(MO+1)=DIVER(IF+4,IE,IA)*Y(1,IE,IH)+DVP(MO
1+1)
50 CONTINUE
IF(MO.NE.1) GO TO 60
IF(IDER(2).EQ.1) DVP(2)=DIVER(9,1,IA)*Y(1,1,ID+4)+DVP(2)
C
C USES DIFFERENTIAL EQUATION TO OBTAIN TIME DERIVATIVES AND ERRORS
C
60 IF(IDER(MO).EQ.1) DVP(MO)=DVP(MO)+DV(MO)
IF(IDER(MO+1).EQ.1) DVP(MO+1)=DVP(MO+1)+DV(MO+1)

```

DFN15000
 DFN15100
 DFN15200
 DFN15300
 DFN15400
 DFN15500
 DFN15600
 DFN15700
 DFN15800
 DFN15900
 DFN16000
 DFN16100
 DFN16200
 DFN16300
 DFN16400
 DFN16500
 DFN16600
 DFN16700
 DFN16800
 DFN16900
 DFN17000
 DFN17100
 DFN17200
 DFN17300
 DFN17400
 DFN17500
 DFN17600
 DFN17700
 DFN17800
 DFN17900
 DFN18000
 DFN18100
 DFN18200
 DFN18300
 DFN18400
 DFN18500
 DFN18600
 DFN18700
 DFN18800
 DFN18900
 DFN19000
 DFN19100
 DFN19200
 DFN19300
 DFN19400
 DFN19500
 DFN19600
 DFN19700
 DFN19800
 DFN19900
 DFN20000
 DFN20100
 DFN20200
 DFN20300
 DFN20400

65	CALL UFUN(MO,X(IA),Y(1,1,IA),T,DVP,ERR(1,IA),5)	DFN20500
	CALL UFUN(MO,X(IA),Y(1,1,IA),T,DV,DY(1,IA),5)	DFN20600
	DO 100 IB=1,MO	DFN20700
	IF(L2.EQ.1) GO TO 100	DFN20800
	ERR(IB,IA) = DY(IB,IA) - ERR(IB,IA)	DFN20900
100	CONTINUE	DFN21000
	IF(L3.EQ.0) GO TO 105	DFN21100
	DO 103 IA=1,MO	DFN21200
	Y(1,IA,1)=SV(IA)	DFN21300
103	Y(1,IA,N)=SV(IA+3)	DFN21400
105	CONTINUE	DFN21500
	IF (L .EQ. 0) RETURN	DFN21600
	X1 = X(1)	DFN21700
	XN = X(N)	DFN21800
	DO 110 I = 1,5	DFN21900
	I1 = I + 1	DFN22000
	I2 = N - I	DFN22100
	XT(I) = X(I1)	DFN22200
	XS(I) = X(I2)	DFN22300
	DO 110 J = 1,MO	DFN22400
	YT(J,I) = DY(J,I1)	DFN22500
	YS(J,I) = DY(J,I2)	DFN22600
110	CONTINUE	DFN22700
	CALL DERIV(XT,YT,IDER,MO,0,1,0,MO,X1,D,XI)	DFN22800
	CALL DERIV(XS,YS,IDER,MO,0,1,0,MO,XN,ER,XI)	DFN22900
	DO 120 I = 1,MO	DFN23000
	DY(I,1) = D(I)	DFN23100
	DY(I,N) = ER(I)	DFN23200
120	CONTINUE	DFN23300
	RETURN	DFN23400
C		DFN23500
C	SETS PARAMETERS AND FORMS DERIVATIVES FOR HERMITE	DFN23600
C		DFN23700
70	DV(1)=Y(1,2,IG)	DFN23800
	DVP(1)=DV(1)	DFN23900
	IF(IDER(2).EQ.C) GO TO 75	DFN24000
	DVP(2)=0.	DFN24100
	DV(2)=2.*(Y(1,1,IB)-2.*Y(1,1,IG)+Y(1,1,IC))/XIS-(Y(1,2,IB)-Y(1,2,IC))/(2.*XI)	DFN24200
75	IF(IDER(3).EQ.0) GO TO 40	DFN24300
	DVP(3)=0.	DFN24400
	DV(3)=7.500000C*(Y(1,1,IB)-Y(1,1,IC))/XIC-1.500000C*(Y(1,2,IB)	DFN24500
	1 +8.*Y(1,2,IG) + Y(1,2,IC))/XIS	DFN24600
	GO TO 40	DFN24700
C		DFN24800
C	SETS PARAMETERS AND FORMS DERIVATIVES FOR TRIMITE	DFN24900
C		DFN25000
80	DV(1)=Y(1,2,IG)	DFN25100
	DVP(1)=DV(1)	DFN25200
	DV(2)=Y(1,3,IG)	DFN25300
	DVP(2)=DV(2)	DFN25400
	IF(IDER(3).EQ.0) GO TO 85	DFN25500
	DVP(3)=0.	DFN25600
	DV(3)=13.12500C*(Y(1,1,IB)-Y(1,1,IC))/XIC-(4.125000C*(Y(1,2,IB)	DFN25700
	1 +Y(1,2,IC))+18.000000C*Y(1,2,IG))/XIS+.37500000C*(Y(1,3,IB) -	DFN25800
		DFN25900

```

      2 Y(1,3,IC))/XI
85  IF(IDER(4).EQ.0) GO TO 40
      DVP(4)=0.
      DV(4)=72.000000*(Y(1,1,IB)-2.*Y(1,1,IG)+Y(1,1,IC))/XIF-19.500000*
1(Y(1,2,IB)-Y(1,2,IC))/XIC+1.500000*(Y(1,3,IB)-24.000000*Y(1,3,IG)
2 +Y(1,3,IC))/XIS
      GO TO 40
      END

```

DFN26000
DFN26100
DFN26200
DFN26300
DFN26400
DFN26500
DFN26600
DFN26700

```

SUBROUTINE DIFSUB(N,T,Y,SAVE,H,HMIN,HMAX,EPS,YMAX,ERROR,KFLAG,
1JSTART,MAXDER,PSAVE,K7,HMY,X,DIVER,INT,IDER,ERR,MO,IDOUB,IQQQC,
2HSAV,PEQ,ERORA,ITPSC)
IMPLICIT REAL * 8(A-H,P-Z)
REAL * 4 PEP SH,PR1,PR2,PR3,PERTST

```

DSB10000
DSB10100
DSB10200
DSB10300
DSB10400
DSB10500
DSB10600
DSB10700
DSB10800
DSB10900
DSB11000
DSB11100
DSB11200
DSB11300
DSB11400
DSB11500
DSB11600
DSB11700
DSB11800
DSB11900
DSB12000
DSB12100
DSB12200
DSB12300
DSB12400
DSB12500
DSB12600
DSB12700
DSB12800
DSB12900
DSB13000
DSB13100
DSB13200
DSB13300
DSB13400
DSB13500
DSB13600
DSB13700
DSB13800
DSB13900
DSB14000
DSB14100

```

C
C THE PARAMETERS TO THE SUBROUTINE DIFSUB HAVE
C THE FOLLOWING MEANINGS..
C
C N      THE NUMBER OF FIRST ORDER DIFFERENTIAL EQUATIONS. N
C        MAY BE DECREASED ON LATER CALLS IF THE NUMBER OF
C        ACTIVE EQUATIONS REDUCES, BUT IT MUST NOT BE
C        INCREASED WITHOUT CALLING WITH JSTART = 0.
C T      THE INDEPENDENT VARIABLE.
C Y      AN 8 BY N ARRAY CONTAINING THE DEPENDENT VARIABLES AND
C        THEIR SCALED DERIVATIVES. Y(J+1,I) CONTAINS
C        THE J-TH DERIVATIVE OF Y(I) SCALED BY
C        H**J/FACTORIAL(J) WHERE H IS THE CURRENT
C        STEP SIZE. ONLY Y(1,I) NEED BE PROVIDED BY
C        THE CALLING PROGRAM ON THE FIRST ENTRY.
C        IF IT IS DESIRED TO INTERPOLATE TO NON- MESH POINTS
C        THESE VALUES CAN BE USED. IF THE CURRENT STEP SIZE
C        IS H AND THE VALUE AT T + E IS NEEDED, FORM
C        S = E/H, AND THEN COMPUTE
C
C          NQ
C          Y(I)(T+E) = SUM Y(J+1,I)*S**J
C                      J=0
C
C SAVE   A BLOCK OF AT LEAST 12*N FLOATING POINT LOCATIONS
C        USED BY THE SUBROUTINES.
C
C H      THE STEP SIZE TO BE ATTEMPTED ON THE NEXT STEP.
C        H MAY BE ADJUSTED UP OR DOWN BY THE PROGRAM
C        IN ORDER TO ACHEIVE AN ECONOMICAL INTEGRATION.
C        HOWEVER, IF THE H PROVIDED BY THE USER DOES
C        NOT CAUSE A LARGER ERROR THAN REQUESTED, IT
C        WILL BE USED. TO SAVE COMPUTER TIME, THE USER IS
C        ADVISED TO USE A FAIRLY SMALL STEP FOR THE FIRST
C        CALL. IT WILL BE AUTOMATICALLY INCREASED LATER.
C
C HMIN   THE MINIMUM STEP SIZE THAT WILL BE USED FOR THE
C        INTEGRATION. NOTE THAT ON STARTING THIS MUST BE
C        MUCH SMALLER THAN THE AVERAGE H EXPECTED SINCE
C        A FIRST ORDER METHOD IS USED INITIALLY.
C
C HMAX   THE MAXIMUM SIZE TO WHICH THE STEP WILL BE INCREASED

```

EPS	THE ERROR TEST CONSTANT. SINGLE STEP ERROR ESTIMATES DIVIDED BY YMAX(I) MUST BE LESS THAN THIS IN THE EUCLIDEAN NORM. THE STEP AND/OR ORDER IS ADJUSTED TO ACHIEVE THIS.	DSB14200
MF	THE METHOD INDICATOR. THE FOLLOWING ARE ALLOWED..	DSB14300
	0 AN ADAMS PREDICTOR CORRECTOR IS USED.	DSB14400
	1 A MULTI-STEP METHOD SUITABLE FOR STIFF SYSTEMS IS USED. IT WILL ALSO WORK FOR NON STIFF SYSTEMS. HOWEVER THE USER MUST PROVIDE A SUBROUTINE PEDERV WHICH EVALUATES THE PARTIAL DERIVATIVES OF THE DIFFERENTIAL EQUATIONS WITH RESPECT TO THE Y'S. THIS IS DONE BY CALL PEDERV(I,Y,PW,M). PW IS AN N BY N ARRAY WHICH MUST BE SET TO THE PARTIAL OF THE I-TH EQUATION WITH RESPECT TO THE J DEPENDENT VARIABLE IN PW(I,J). PW IS ACTUALLY STORED IN AN M BY M ARRAY WHERE M IS THE VALUE OF N USED ON THE FIRST CALL TO THIS PROGRAM.	DSB14500
	2 THE SAME AS CASE 1, EXCEPT THAT THIS SUBROUTINE COMPUTES THE PARTIAL DERIVATIVES BY NUMERICAL DIFFERENCING OF THE DERIVATIVES. HENCE PEDERV IS NOT CALLED.	DSB14600
YMAX	AN ARRAY OF N LOCATIONS WHICH CONTAINS THE MAXIMUM OF EACH Y SEEN SO FAR. IT SHOULD NORMALLY BE SET TO 1 IN EACH COMPONENT BEFORE THE FIRST ENTRY. (SEE THE DESCRIPTION OF EPS.)	DSB14700
ERROR	AN ARRAY OF N ELEMENTS WHICH CONTAINS THE ESTIMATED ONE STEP ERROR IN EACH COMPONENT.	DSB14800
KFLAG	A COMPLETION CODE WITH THE FOLLOWING MEANINGS..	DSB14900
	+1 THE STEP WAS SUCCESSFUL	DSB15000
	-1 THE STEP WAS TAKEN WITH H = HMIN, BUT THE REQUESTED ERROR WAS NOT ACHIEVED.	DSB15100
	-2 THE MAXIMUM ORDER SPECIFIED WAS FOUND TO BE TOO LARGE.	DSB15200
	-3 CORRECTOR CONVERGENCE COULD NOT BE ACHIEVED FOR H .GT. HMIN.	DSB15300
	-4 THE REQUESTED ERROR IS SMALLER THAN CAN BE HANDLED FOR THIS PROBLEM.	DSB15400
JSTART	AN INPUT INDICATOR WITH THE FOLLOWING MEANINGS..	DSB15500
	-1 REPEAT THE LAST STEP WITH A NEW H	DSB15600
	0 PERFORM THE FIRST STEP. THE FIRST STEP MUST BE DONE WITH THIS VALUE OF JSTART SO THAT THE SUBROUTINE CAN INITIALIZE ITSELF.	DSB15700
	+1 TAKE A NEW STEP CONTINUING FROM THE LAST. JSTART IS SET TO NQ, THE CURRENT ORDER OF THE METHOD AT EXIT. NQ IS ALSO THE ORDER OF THE MAXIMUM DERIVATIVE AVAILABLE.	DSB15800
MAXDER	THE MAXIMUM DERIVATIVE THAT SHOULD BE USED IN THE METHOD. SINCE THE ORDER IS EQUAL TO THE HIGHEST DERIVATIVE USED, THIS RESTRICTS THE ORDER. IT MUST BE LESS THAN 8 FOR ADAMS AND 7 FOR STIFF METHODS.	DSB15900
		DSB16000
		DSB16100
		DSB16200
		DSB16300
		DSB16400
		DSB16500
		DSB16600
		DSB16700
		DSB16800
		DSB16900
		DSB17000
		DSB17100
		DSB17200
		DSB17300
		DSB17400
		DSB17500
		DSB17600
		DSB17700
		DSB17800
		DSB17900
		DSB18000
		DSB18100
		DSB18200
		DSB18300
		DSB18400
		DSB18500
		DSB18600
		DSB18700
		DSB18800
		DSB18900
		DSB19000
		DSB19100
		DSB19200
		DSB19300
		DSB19400
		DSB19500
		DSB19600

	IRET1 = 1	DSB25200
	GO TO 750	DSB25300
130	NQOLD = NQ	DSB25400
	TOLD = T	DSB25500
	RACUM = 1.0	DSB25600
	IF(JSTART.GT.0.AND.IQQQC.NE.1) GO TO 250	DSB25700
	GO TO 170	DSB25800
140	IF (JSTART.EQ.-1) GO TO 160	DSB25900
C		DSB26000
C	ON THE FIRST CALL, THE ORDER IS SET TO 1 AND THE INITIAL	DSB26100
C	DERIVATIVES ARE CALCULATED.	DSB26200
C		DSB26300
	NQ = 1	DSB26400
141	N3 = N	DSB26500
	NMO = N - MO	DSB26600
	N1=N*7	DSB26700
	L3L = 9*N	DSB26800
	N2 = N1 + 1	DSB26900
	N4 = N**2	DSB27000
	N5 = N1 + N	DSB27100
	N6 = N5 + 1	DSB27200
	NY1 = N*3	DSB27300
	NY2=NY1+1	DSB27400
	NY5=NY1+N	DSB27500
	NY6=NY5+1	DSB27600
	IF(JSTART.GT.0) GO TO 100	DSB27700
	JRET=1	DSB27800
	A(1) =-.50000000000000	DSB27900
	CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N2,1),1,0,0)	DSB28000
	EPSN = .16700 * EPS/DFLOAT(N)	DSB28100
	DO 144 I=1,N	DSB28200
	N11 = N1 + I	DSB28300
	Y(2,I) = SAVE(N11,1)*H	DSB28400
	ERROR(I) = EPSN	DSB28500
144	CONTINUE	DSB28600
	CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N2,1),0,1,1)	DSB28700
	GO TO 310	DSB28800
142	CONTINUE	DSB28900
	CALL DIVERR(PEQ,N,ERROR,ERR,HSAB(NY2,1),HMY,PERTST(NQ,1),K7,1,MO	DSB29000
	1,X)	DSB29100
	JRET=2	DSB29200
	DO 150 I = 1,N	DSB29300
	ERROR(I) = 0.0	DSB29400
	NY11=NY1+I	DSB29500
	HMY(2,I)=HSAB(NY11,1)*H	DSB29600
150	CONTINUE	DSB29700
	HNEW = H	DSB29800
	K = 2	DSB29900
	GO TO 100	DSB30000
C		DSB30100
C	REPEAT LAST STEP BY RESTORING SAVED INFORMATION.	DSB30200
C		DSB30300
160	IF (NQ.EQ.NQOLD) JSTART = 1	DSB30400
	T=TOLD	DSB30500
	NQ = NQOLD	DSB30600

```

      K = NQ + 1
      GO TO 120

C
C SET THE COEFFICIENTS THAT DETERMINE THE ORDER AND THE METHOD
C TYPE. CHECK FOR EXCESSIVE ORDER. THE LAST TWO STATEMENTS OF
C THIS SECTION SET IWEVAL .GT. J IF PW IS TO BE RE-EVALUATED
C BECAUSE OF THE ORDER CHANGE, AND THEN REPEAT THE INTEGRATION
C STEP IF IT HAS NOT YET BEEN DONE (IRET = 1) OR SKIP TO A FINAL
C SCALING BEFORE EXIT IF IT HAS BEEN COMPLETED (IRET = 2).
C
170 IF(NQ .GT. 4) GO TO 190
      GO TO (221,222,223,224),NQ
190 KFLAG = -2
      RETURN

C
C THE FOLLOWING COEFFICIENTS SHOULD BE DEFINED TO THE MAXIMUM
C ACCURACY PERMITTED BY THE MACHINE. THEY ARE IN THE ORDER USED..
C
C      -1/2
C      -1/2,-1/2
C      -1/2,-5/8,-1/8
C      -1/2,-47/72,-1/6,-1/72
C
221 A(1)=-.5000000000
      GO TO 230
222 A(1)=-.5000000000
      A(3)=A(1)
      GO TO 230
223 A(1)=-.5000000000
      A(3)=-.6250000000
      A(4)=-.125000000000
      GO TO 230
224 A(1)=-.5000000000
      A(3)=-.6527777777777778
      A(4)=-.1666666666666667
      A(5)=-.01388888888888889
230 K = NQ+1
      IDOUB = K
231 CONTINUE
      ENQ2 = .5/FLOAT(NQ + 1)
      ENQ3 = .5/FLOAT(NQ + 2)
      ENQ1 = 0.5/FLOAT(NQ)
      PEP SH = EPS
      EUP = (PERTST(NQ,2)*PEPSH)**2
      E = (PERTST(NQ,1)*PEPSH)**2
      EDWN=(PERTST(NQ,3)*PEPSH)**2
      IF (EDWN.EQ.0) GO TO 780
      BND = EPS*ENQ3/DFLOAT(N)
      IF(ITPSC .EQ. 1 ) GO TO 2
240 IWEVAL = MF
      GO TO ( 250 , 680 ),IRET

C
C THIS SECTION COMPUTES THE PREDICTED VALUES BY EFFECTIVELY
C MULTIPLYING THE SAVED INFORMATION BY THE PASCAL TRIANGLE
C MATRIX.

```

DSB30700
 DSB30800
 DSB30900
 DSB31000
 DSB31100
 DSB31200
 DSB31300
 DSB31400
 DSB31500
 DSB31600
 DSB31700
 DSB31800
 DSB31900
 DSB32000
 DSB32100
 DSB32200
 DSB32300
 DSB32400
 DSB32500
 DSB32600
 DSB32700
 DSB32800
 DSB32900
 DSB33000
 DSB33100
 DSB33200
 DSB33300
 DSB33400
 DSB33500
 DSB33600
 DSB33700
 DSB33800
 DSB33900
 DSB34000
 DSB34100
 DSB34200
 DSB34300
 DSB34400
 DSB34500
 DSB34600
 DSB34700
 DSB34800
 DSB34900
 DSB35000
 DSB35100
 DSB35200
 DSB35300
 DSB35400
 DSB35500
 DSB35600
 DSB35700
 DSB35800
 DSB35900
 DSB36000
 DSB36100

C			DSB36200
250	T = T + H		DSB36300
	DO 260 J = 2,K		DSB36400
	DO 260 J1 = J,K		DSB36500
	J2 = K - J1 + J - 1		DSB36600
	DO 260 I = 1,N		DSB36700
	IF(J .EQ. 2 .AND. J1 .EQ. 2) HMY(1,I) = HMY(1,I) + HMY(2,I)		DSB36800
260	Y(J2,I) = Y(J2,I) + Y(J2+1,I)		DSB36900
C			DSB37000
C	UP TO 3 CORRECTOR ITERATIONS ARE TAKEN. CONVERGENCE IS TESTED		DSB37100
C	BY REQUIRING CHANGES TO BE LESS THAN BND WHICH IS DEPENDENT ON		DSB37200
C	THE ERROR TEST CONSTANT.		DSB37300
C	THE SUM OF THE CORRECTIONS IS ACCUMULATED IN THE ARRAY		DSB37400
C	ERROR(I). IT IS EQUAL TO THE K-TH DERIVATIVE OF Y MULTIPLIED		DSB37500
C	BY H**K/(FACTORIAL(K-1)*A(K)), AND IS THEREFORE PROPORTIONAL		DSB37600
C	TO THE ACTUAL ERRORS TO THE LOWEST POWER OF H PRESENT. (H**K)		DSB37700
			DSB37800
	DO 270 I = 1,N		DSB37900
270	ERROR(I) = 0.0		DSB38000
	L1=3		DSB38100
	DO 430 L = 1,L1		DSB38200
	CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N2,1),0,0,0)		DSB38300
C			DSB38400
C	IF THERE HAS BEEN A CHANGE OF ORDER OR THERE HAS BEEN TROUBLE		DSB38500
C	WITH CONVERGENCE, PW IS RE-EVALUATED PRIOR TO STARTING THE		DSB38600
C	CORRECTOR ITERATION IN THE CASE OF STIFF METHODS. IWEVAL IS		DSB38700
C	THEN SET TO -1 AS AN INDICATOR THAT IT HAS BEEN DONE.		DSB38800
			DSB38900
	IF(IWEVAL.LT.1) GO TO 370		DSB39000
	GO TO 310		DSB39100
C			DSB39200
C	SETS THE JACOBIAN FOR USE WITH ASYMPTOTIC ERROR EQUATION		DSB39300
			DSB39400
271	DO 272 I=1,K7		DSB39500
	DO 272 J=1,N		DSB39600
272	PEQ(I,J)=PSAVE(I,J)		DSB39700
C			DSB39800
C	PREPARES TO DECOMPOSE THE MATRIX PSAVE		DSB39900
			DSB40000
	IF(JRET.EQ.1) GO TO 142		DSB40100
	IF(JRET.EQ.3) GO TO 721		DSB40200
273	R = A(1)*H		DSB40300
	DO 280 I = 1,K7		DSB40400
	K8=K2+2-I		DSB40500
	K8= MAX0(1,K8)		DSB40600
	K9=N+K2+1-I		DSB40700
	K9=MIN0(N,K9)		DSB40800
	DO 280 J=K8,K9		DSB40900
	IF(J .LE. MO .OR. J .GT. NMO) GO TO 275		DSB41000
	PSAVE(I,J) = PSAVE(I,J)*R		DSB41100
	GO TO 280		DSB41200
275	PSAVE(I,J) = PSAVE(I,J) * A(1)		DSB41300
280	CONTINUE		DSB41400
290	DO 300 I=1,N		DSB41500
	IF(I .LE. MO .OR. I .GT. NMO) GO TO 300		DSB41600

```

PSAVE(K3,I) = 1.0 + PSAVE(K3,I)
300 CONTINUE
    IWEVAL = -1
    DO 308 I = 1,MO
        J = K2-I+2
        J3 = 0
        DO 303 J2 = J,K7
            IF(DABS(PSAVE(J2,I)) .LT. 1.0D-20) J3 = J3 + 1
303 CONTINUE
        J = K7-J+1
        IF(J3 .GE. J) PSAVE(K3,I) = 1.0D0
        J3 = 0
        J1 = NMO + I
        J = K3+MO-I
        DO 305 J2 = 1,J
            IF(DABS(PSAVE(J2,J1)) .LT. 1.0D-20) J3 = J3 + 1
305 CONTINUE
        IF(J3 .GE. J) PSAVE(K3,J1) = 1.0D0
308 CONTINUE
        CALL DECOMP(N,PSAVE,J1,K7)
C
C CALCULATE THE JACOBIAN BY NUMERICAL DIFFERENCING
C
    IF(J1.GT.0) GO TO 370
    WRITE(6,11112)
11112 FORMAT(' MATRIX PROBLEM')
    GO TO 440
310 DO 320 I = 1,N
320     SAVE(6,I) = Y(1,I)
    DO 340 J = 1,N
        R = EPS*DMAX1(EPS,DABS(SAVE(6,J)))
        Y(1,J) = Y(1,J) + R
    D=1./R
    CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N6,1),0,1,1)
    DO 335 I = 1,K7
        JI=J-K3+I
        IF(JI.LE.0) GO TO 335
        IF(JI.GT.N) GO TO 340
        I1=K7+1-I
        N12=N5+JI
        N13=N1+JI
330 PSAVE(I1,JI)=(SAVE(N12,1)-SAVE(N13,1))*D
335 CONTINUE
340     Y(1,J) = SAVE(6,J)
        GO TO 271
350 CONTINUE
370 DO 380 I = 1,N
        N11 = N5 + I
        N12 = N1 + I
        SAVE(N11,1) = Y(2,I) - SAVE(N12,1)*H
        IF(I .LE. MO .OR. I .GT. NMO) SAVE(N11,1) = -SAVE(N12,1)
380 CONTINUE
C
C COMPLETE THE GAUSSIAN ELIMINATION
C

```

DSB41700
DSB41800
DSB41900
DSB42000
DSB42100
DSB42200
DSB42300
DSB42400
DSB42500
DSB42600
DSB42700
DSB42800
DSB42900
DSB43000
DSB43100
DSB43200
DSB43300
DSB43400
DSB43500
DSB43600
DSB43700
DSB43800
DSB43900
DSB44000
DSB44100
DSB44200
DSB44300
DSB44400
DSB44500
DSB44600
DSB44700
DSB44800
DSB44900
DSB45000
DSB45100
DSB45200
DSB45300
DSB45400
DSB45500
DSB45600
DSB45700
DSB45800
DSB45900
DSB46000
DSB46100
DSB46200
DSB46300
DSB46400
DSB46500
DSB46600
DSB46700
DSB46800
DSB46900
DSB47000
DSB47100

```

CALL SOLVE(N,PSAVE,SAVE(N5+1,1),SAVE,K7,7,6)
410 NT = N
C
C PERFORM THE CORRECTION STEP
C
DO 420 I = 1,N
Y(1,I) = Y(1,I) + A(1)*SAVE(6,I)
Y(2,I) = Y(2,I) - SAVE(6,I)
ERROR(I) = ERROR(I) + SAVE(6,I)
IF (DABS(SAVE(6,I)).LE.(BND*YMAX(I))) NT = NT - 1
420 CONTINUE
CALL DIVERR(PEQ,N,ERROR,ERR,NSAV(NY2,1),HMY,PERTST(NQ,1),K7,0,MD
1,X)
DO 425 I=1,N
NY11=NY5+I
NY12=NY1+I
NSAV(NY11,1)=HMY(2,I)-NSAV(NY12,1)*H
IF(I.LE.MO.OR.I.GT.NMO) NSAV(NY11,1) = - NSAV(NY12,1)
425 CONTINUE
C
C SOLVE THE ASYMPTOTIC ERROR EQUATION
C
CALL SOLVE(N,PSAVE,NSAV(NY5+1,1),NSAV,K7,3,3)
DO 427 I=1,N
HMY(1,I)=HMY(1,I)+A(1)*NSAV(3,I)
427 HMY(2,I)=HMY(2,I)- NSAV(3,I)
IF (NT.LE.0) GO TO 490
430 CONTINUE
C
C THE CORRECTOR ITERATION FAILED TO CONVERGE IN 3 TRIES. VARIOUS
C POSSIBILITIES ARE CHECKED FOR. IF H IS ALREADY HMIN AND
C THIS IS EITHER ADAMS METHOD OR THE STIFF METHOD IN WHICH THE
C MATRIX PW HAS ALREADY BEEN RE-EVALUATED, A NO CONVERGENCE EXIT
C IS TAKEN. OTHERWISE THE MATRIX PW IS RE-EVALUATED AND/OR THE
C STEP IS REDUCED TO TRY AND GET CONVERGENCE.
C
440 T=TOLD
IF ((H.LE.(HMIN*1.00001)).AND.((IWEVAL - MTP).LT.-1)) GO TO 460
IF ((MF.EQ.0).OR.(IWEVAL.NE.0)) RACUM = RACUM*0.2500
IWEVAL = MF
IRET1 = 2
GO TO 750
460 KFLAG = -3
470 DO 480 I = 1,N
HMY(1,I) = NSAV(1,I)
HMY(2,I) = NSAV(2,I)
DO 480 J = 1,K
480 Y(J,I) = SAVE(J,I)
H = HOLD
NQ = NQOLD
JSTART = NQ
RETURN
C
C THE CORRECTOR CONVERGED AND CONTROL IS PASSED TO STATEMENT 520
C IF THE ERROR TEST IS O.K., AND TO 540 OTHERWISE.

```

DSB47200
DSB47300
DSB47400
DSB47500
DSB47600
DSB47700
DSB47800
DSB47900
DSB48000
DSB48100
DSB48200
DSB48300
DSB48400
DSB48500
DSB48600
DSB48700
DSB48800
DSB48900
DSB49000
DSB49100
DSB49200
DSB49300
DSB49400
DSB49500
DSB49600
DSB49700
DSB49800
DSB49900
DSB50000
DSB50100
DSB50200
DSB50300
DSB50400
DSB50500
DSB50600
DSB50700
DSB50800
DSB50900
DSB51000
DSB51100
DSB51200
DSB51300
DSB51400
DSB51500
DSB51600
DSB51700
DSB51800
DSB51900
DSB52000
DSB52100
DSB52200
DSB52300
DSB52400
DSB52500
DSB52600

```

C IF THE STEP IS O.K. IT IS ACCEPTED. IF IDOUB HAS BEEN REDUCED
C TO ONE, A TEST IS MADE TO SEE IF THE STEP CAN BE INCREASED
C AT THE CURRENT ORDER OR BY GOING TO ONE HIGHER OR ONE LOWER.
C SUCH A CHANGE IS ONLY MADE IF THE STEP CAN BE INCREASED BY AT
C LEAST 1.1. IF NO CHANGE IS POSSIBLE IDOUB IS SET TO 10 TO
C PREVENT FURTHER TESTING FOR 10 STEPS.
C IF A CHANGE IS POSSIBLE, IT IS MADE AND IDOUB IS SET TO
C NQ + 1 TO PREVENT FURTHER TESING FOR THAT NUMBER OF STEPS.
C IF THE ERROR WAS TOO LARGE, THE OPTIMUM STEP SIZE FOR THIS OR
C LOWER ORDER IS COMPUTED, AND THE STEP RETRIED. IF IT SHOULD
C FAIL TWICE MORE IT IS AN INDICATION THAT THE DERIVATIVES THAT
C HAVE ACCUMULATED IN THE Y ARRAY HAVE ERRORS OF THE WRONG ORDER
C SO THE FIRST DERIVATIVES ARE RECOMPUTED AND THE ORDER IS SET
C TO 1.
DSB52700
DSB52800
DSB52900
DSB53000
DSB53100
DSB53200
DSB53300
DSB53400
DSB53500
DSB53600
DSB53700
DSB53800
DSB53900
DSB54000
DSB54100
DSB54200
DSB54300
DSB54400
DSB54500
DSB54600
DSB54700
DSB54800
DSB54900
DSB55000
DSB55100
DSB55200
DSB55300
DSB55400
DSB55500
DSB55600
DSB55700
DSB55800
DSB55900
DSB56000
DSB56100
DSB56200
DSB56300
DSB56400
DSB56500
DSB56600
DSB56700
DSB56800
DSB56900
DSB57000
DSB57100
DSB57200
DSB57300
DSB57400
DSB57500
DSB57600
DSB57700
DSB57800
DSB57900
DSB58000
DSB58100

490 D = 0.0
DO 500 I = 1,N
500 D = D + (ERROR(I)/YMAX(I))**2
IWEVAL = 0
IF (D.GT.E) GO TO 540
IF (K.LT.3) GO TO 520

C
C STEP SUCCEEDED, UPDATE REMAINING VECTORS
C
DO 510 J = 3,K
DO 510 I = 1,N
510 Y(J,I) = Y(J,I) + A(J)*ERROR(I)
520 KFLAG = +1
HNEW = H
IF (IDOUB.LE.1) GO TO 550
IDOUB = IDOUB - 1
IF (IDOUB.GT.1) GO TO 700
DO 530 I = 1,N
530 SAVE(7 ,I) = ERROR(I)
GO TO 700

C
C ERROR TOO LARGE, REDUCE FAILURE FLAG AND TRY AGAIN
C
540 KFLAG = KFLAG - 2
IF (H.LE.(HMIN*1.00001)) GO TO 740
T = TOLD
IF (KFLAG.LE.-11) GO TO 720

C
C CHECKS TO SEE IF CHANGES SHOULD BE MADE IN STEP SIZE OR ORDER
C
550 PR2 = (D/E)**ENQ2*1.2
PR3 = 1.E+20
IF ((NQ.GE.MAXDER).OR.(KFLAG.LE.-1)) GO TO 570
D = 0.C
DO 560 I = 1,N
560 D = D + ((ERROR(I) - SAVE(7 ,I))/YMAX(I))**2
PR3 = (D/EUP)**ENQ3*1.4
570 PR1 = 1.E+20
IF (NQ.LE.1) GO TO 590
D = 0.0

```

DO 580 I = 1,N	DSB58200
580 D = D + (Y(K,I)/YMAX(I))*2	DSB58300
PR1 = (D/EDWN)**ENQ1*1.3	DSB58400
590 CONTINUE	DSB58500
IF (PR2.LE.PR3) GO TO 650	DSB58600
IF (PR3.LT.PR1) GO TO 660	DSB58700
600 R = 1.0/AMAX1(PR1,1.E-4)	DSB58800
NEWQ = NQ - 1	DSB58900
610 IDOUB = 10	DSB59000
IF ((KFLAG.EQ.1).AND.(R.LT.(1.1))) GO TO 700	DSB59100
IF (NEWQ.LE.NQ) GO TO 630	DSB59200
C	DSB59300
C COMPUTE AN ADDITIONAL SCALED DERIVATIVE IF ORDER IS INCREASED	DSB59400
C	DSB59500
DO 620 I = 1,N	DSB59600
620 Y(NEWQ+1,I) = ERROR(I)*A(K)/DFLOAT(K)	DSB59700
630 K = NEWQ + 1	DSB59800
IF (KFLAG.EQ.1) GO TO 670	DSB59900
RACUM = RACUM*R	DSB60000
IRET1 = 3	DSB60100
GO TO 750	DSB60200
640 IF (NEWQ.EQ.NQ) GO TO 250	DSB60300
NQ = NEWQ	DSB60400
GO TO 170	DSB60500
650 IF (PR2.GT.PR1) GO TO 600	DSB60600
NEWQ = NQ	DSB60700
R = 1.0/AMAX1(PR2,1.E-4)	DSB60800
GO TO 610	DSB60900
660 R = 1.0/AMAX1(PR3,1.E-4)	DSB61000
NEWQ = NQ + 1	DSB61100
GO TO 610	DSB61200
670 IRET = 2	DSB61300
R = DMIN1(R,HMAX/DABS(H))	DSB61400
H = H*R	DSB61500
HNEW = H	DSB61600
IF (NQ.EQ.NEWQ) GO TO 680	DSB61700
NQ = NEWQ	DSB61800
GO TO 170	DSB61900
680 R1 = 1.0	DSB62000
DO 690 J = 2,K	DSB62100
R1 = R1*R	DSB62200
DO 690 I = 1,N	DSB62300
IF (J.EQ.2) HMY(J,I) = HMY(J,I) *R1	DSB62400
690 Y(J,I) = Y(J,I)*R1	DSB62500
IDOUB = K	DSB62600
700 DO 710 I = 1,N	DSB62700
YMAX(I) = DMAX1(YMAX(I),DABS(Y(1,I)))	DSB62800
710 ERORA(I)=(HMY(1,I)-HSAV(1,I))/YMAX(I)	DSB62900
JSTART = NQ	DSB63000
RETURN	DSB63100
720 IF (NQ.EQ.1) GO TO 780	DSB63200
CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N2,1),1,0,0)	DSB63300
JRET=3	DSB63400
EPSN = .167DO * EPS/DFLOAT(N)	DSB63500
DO 722 I=1,N	DSB63600

	R = H/HOLD	DSB63700
	Y(1,I) = SAVE(1,I)	DSB63800
	N11 = N1 + I	DSB63900
	SAVE(2,I) = HOLD*SAVE(N11,1)	DSB64000
	Y(2,I) = SAVE(2,I) * R	DSB64100
	ERROR(I) = EPSN	DSB64200
722	CONTINUE	DSB64300
	CALL DIFFUN(X,Y,T,DIVER,MO,INT,N14,IDER,ERR,SAVE(N2,1),0,1,1)	DSB64400
	GO TO 310	DSB64500
721	CONTINUE	DSB64600
	JRET=2	DSB64700
	CALL DIVERR(PEQ,N,ERROR,ERR,HSAB(NY2,1),HMY,PERTST(NQ,1),K7,1,MO	DSB64800
	1,X)	DSB64900
	R = H/HOLD	DSB65000
	DO 730 I = 1,N	DSB65100
	ERROR(I) = 0.0	DSB65200
	HMY(1,I)=HSAB(1,I)	DSB65300
	NY11=NY1+I	DSB65400
	HSAB(2,I)=HOLD*HSAB(NY11,1)	DSB65500
	HMY(2,I)=HSAB(2,I)*R	DSB65600
730	CONTINUE	DSB65700
	NQ = 1	DSB65800
	KFLAG = 1	DSB65900
	GO TO 170	DSB66000
740	KFLAG = -1	DSB66100
	HNEW = H	DSB66200
	JSTART = NQ	DSB66300
	RETURN	DSB66400
C		DSB66500
C	THIS SECTION SCALES ALL VARIABLES CONNECTED WITH H AND RETURNS	DSB66600
C	TO THE ENTERING SECTION.	DSB66700
750	RACUM = DMAX1(DABS(HMIN/HOLD),RACUM)	DSB66800
	RACUM = DMIN1(RACUM,DABS(HMAX/HOLD))	DSB66900
	R1 = 1.0	DSB67000
	DO 760 J = 2,K	DSB67100
	R1 = R1*RACUM	DSB67200
	DO 760 I = 1,N	DSB67300
	IF(J.EQ. 2) HMY(J,I) = HSAB(J,I) * R1	DSB67400
760	Y(J,I) = SAVE(J,I)*R1	DSB67500
	H = HOLD*RACUM	DSB67600
	DO 770 I = 1,N	DSB67700
	HMY(1,I)=HSAB(1,I)	DSB67800
770	Y(1,I) = SAVE(1,I)	DSB67900
	IDOUB = K	DSB68000
	GO TO (130 , 250 , 640),IRET1	DSB68100
780	KFLAG = -4	DSB68200
	GO TO 470	DSB68300
	END	DSB68400
		DSB68500

SUBROUTINE DIVERR(PEQ,N,ERROR,ERR,DY,Y,PERTST,K7,M,MO,X)

DVR10000

```

C
C THE PURPOSE OF THIS SUBPROGRAM IS TO CALCULATE THE TIME
C DERIVATIVE OF THE ASSYMPTOTIC ERROR EQUATION
C
C THE VARIABLES IN THIS SUBPROGRAM ARE
C   PEQ      PARTIALS OF THE DIFFERENTIAL EQUATIONS
C             WRT THE VARIABLES
C   DY       THE TIME DERIVATIVE VECTOR OF THE ASSYMPTOTIC
C             ERROR EQUATION
C   N        NUMBER OF EQUATIONS
C   ERROR    THE TIME BASED TRUNCATION ERROR VECTOR
C   ERR      THE SPACE BASED TRUNCATION ERROR VECTOR
C   Y        VALUE VECTOR OF THE ASSYMPTOTIC ERROR
C   PERTST   CONSTANTS ASSOCIATED WITH THE ORDER
C   K7       THE NUMBER OF DIAGONALS IN THE JACOBIAN
C
C   IMPLICIT REAL * 8 (A-H,O-Z)
C   REAL * 4 PERTST
C   COMMON /DVR/ DVY(4),COEF(4,3,3),ALPHA(2,4,3),DIV(16,5)
C   DIMENSION DY(1),PEQ(K7,1),ERROR(1),ERR(1),Y(2,1)
C   DIMENSION XT(5),XS(5),YT(3,5),YS(3,5)
C   DIMENSION D(4),ER(4)
C   DIMENSION X(1)
C   K8 = K7/2 + 1
C   K2 = K8 + 1
C   K3 = K8 + N
C   DO 10 IA = 1,N
C   N1 = MAX0(K2-IA,1)
C   N2 = MIN0(K3-IA,K7)
C   K1 = IA - K8
C   DY(IA)=- (ERR(IA)) + (ERROR(IA)/PERTST)
C
C MULTIPLIES THE APPROPRIATE VARIABLES BY THE PARTIALS
C
C   DO 5 IB = N1 , N2
C   DY(IA) = PEQ(IB,IA)*Y(1,K1+IB) + DY(IA)
C 5  CONTINUE
C 10 CONTINUE
C
C THIS SECTION DETERMINES THE VALUES AT THE BOUNDARY
C
C   IF(M.EQ. 0) RETURN
C   L=N/MO
C   NO = 1
C   X1 = X(1)
C   XN = X(L)
C   DO 115 J = 1,MO
C   DO 110 I = 1,5
C   I1 = I + 1
C   I2 = L - I
C   XT(I) = X(I1)
C   XS(I) = X(I2)
C   I1 = I*MO+J
C   YT(1,I) = DY(I1)
C   I2 = MO*(I2-1) + J

```

```

DVR10100
DVR10200
DVR10300
DVR10400
DVR10500
DVR10600
DVR10700
DVR10800
DVR10900
DVR11000
DVR11100
DVR11200
DVR11300
DVR11400
DVR11500
DVR11600
DVR11700
DVR11800
DVR11900
DVR12000
DVR12100
DVR12200
DVR12300
DVR12400
DVR12500
DVR12600
DVR12700
DVR12800
DVR12900
DVR13000
DVR13100
DVR13200
DVR13300
DVR13400
DVR13500
DVR13600
DVR13700
DVR13800
DVR13900
DVR14000
DVR14100
DVR14200
DVR14300
DVR14400
DVR14500
DVR14600
DVR14700
DVR14800
DVR14900
DVR15000
DVR15100
DVR15200
DVR15300
DVR15400
DVR15500

```

```
YS(1,I) = DY(I2)
110 CONTINUE
CALL DERIV(XT,YT,IDER,NO,0,1,0,NO,X1,D,XI)
CALL DERIV(XS,YS,IDER,NO,0,1,0,NO,XN,ER,XI)
DY(J) = D(1)
I1 = MO*(L-1)+J
DY(I1) = ER(1)
115 CONTINUE
RETURN
END
```

DVR15600
DVR15700
DVR15800
DVR15900
DVR16000
DVR16100
DVR16200
DVR16300
DVR16400
DVR16500

APPENDIX B

EXAMPLE 1

	IMPLICIT REAL *8(A-H,Q-Z)	EXA10000
	DIMENSION Y(2,513),YTST(513),M(8),E(513),F(513),TTST(8)	EXA10100
	DATA M/3,7,15,31,63,127,255,511/	EXA10200
	DATA TTST/.125,.25,.5,1.,2.,4.,8.,16./	EXA10300
	RPI=3.141592653589793	EXA10400
	QPI=RPI*RPI	EXA10500
	IP1 = 1	EXA10600
	IP2 = 2	EXA10700
	IP3 = 3	EXA10800
	IP4 = 4	EXA10900
	IP5 = 5	EXA11000
	IP6 = 6	EXA11100
	IP7 = 7	EXA11200
C		EXA11300
C	MAIN LOOP FOR DIFFERENT STEP SIZES	EXA11400
C		EXA11500
	DO 500 IA = 2,7	EXA11600
	RMAX = 0.0	EXA11700
	ID = 1	EXA11800
	KA=1	EXA11900
	KB=2	EXA12000
	T=0.0	EXA12100
	NA=M(IA)	EXA12200
	WRITE(6,903) NA	EXA12300
903	FORMAT('1',I30)	EXA12400
	NB=NA+1	EXA12500
	NUMM=0	EXA12600
	NUMD=0	EXA12700
	NC=NA+2	EXA12800
	HX=1./DFLOAT(NB)	EXA12900
	HT=HX	EXA13000
	ZPI = HX * RPI	EXA13100
C		EXA13200
C	INITILIZE	EXA13300
C		EXA13400
	DO 30 IB = 1,NC	EXA13500
30	Y(KA,IB)=DSIN(ZPI*DFLOAT(IB-1))	EXA13600
	A=.5*HT/(HX*HX)	EXA13700
	B=1.+2.*A	EXA13800
	V=1.-2.*A	EXA13900
	C=A	EXA14000
	NUMM=NUMM+4	EXA14100
	NUMD = NUMD + 1	EXA14200
	E(1)=0.0	EXA14300
	F(1)=0.0	EXA14400
C		EXA14500
C	ONE STEP OF THE INTEGRATION - SOLVE SYSTEM OF EQUATIONS	EXA14600
C		EXA14700
50	Y(KA,1) = 0.0	EXA14800
	DO 100 IB = 2,NB	EXA14900
	CPT=B-C*E(IB-1)	EXA15000
	E(IB)= A/CPT	EXA15100
	D= A*Y(KA,IB-1)+V*Y(KA,IB)+A*Y(KA,IB+1)	EXA15200
100	F(IB)=(D+C*F(IB-1))/CPT	EXA15300

	Y(KB,NC)=0.0	EXA15400
	DO 200 IB=1,NA	EXA15500
	IC=NC-IB	EXA15600
200	Y(KB,IC)=E(IC)*Y(KB,IC+1)+F(IC)	EXA15700
	NUMM=NUMM+6*NA	EXA15800
	NUMD=NUMD+2*NA	EXA15900
	KA=3-KA	EXA16000
	KB=3-KB	EXA16100
	T=T+HT	EXA16200
	IF(T .LT. TTST(ID)) GO TO 50	EXA16300
C		EXA16400
C	OUTPUT SECTION	EXA16500
C		EXA16600
	DO 300 IB=1,NA	EXA16700
	VAL=DEXP(-QPI*T)*DSIN(ZPI*DFLOAT(IB))	EXA16800
	YTST(IB) = Y(KA,IB+1) - VAL	EXA16900
	IF(RMAX.LT. DABS(YTST(IB))) RMAX = DABS(YTST(IB))	EXA17000
300	CONTINUE	EXA17100
	WRITE (6,900) T,HX,A,B	EXA17200
	WRITE(6,904) Y(KA,IP1+1),Y(KA,IP2+1),Y(KA,IP3+1),Y(KA,IP4+1),	EXA17300
	1 Y(KA,IP5+1),Y(KA,IP6+1),Y(KA,IP7+1)	EXA17400
	WRITE(6,904) YTST(IP1),YTST(IP2),YTST(IP3),YTST(IP4),YTST(IP5),	EXA17500
	1YTST(IP6),YTST(IP7)	EXA17600
900	FORMAT(1X,5D26.15)	EXA17700
	WRITE(6,901) NUMM,NUMD	EXA17800
901	FORMAT(1X,3I30)	EXA17900
	WRITE(6,902)	EXA18000
902	FORMAT(' ')	EXA18100
904	FORMAT(1X,7D18.9)	EXA18200
	ID = ID + 1	EXA18300
	IF(T .LT. 16) GO TO 50	EXA18400
	IF(RMAX .LT. .001) CALL EXIT	EXA18500
	IP1 = IP1 * 2	EXA18600
	IP2 = IP2 * 2	EXA18700
	IP3 = IP3 * 2	EXA18800
	IP4 = IP4 * 2	EXA18900
	IP5 = IP5 * 2	EXA19000
	IP6 = IP6 * 2	EXA19100
	IP7 = IP7 * 2	EXA19200
500	CONTINUE	EXA19300
	CALL EXIT	EXA19400
	END	EXA19500

APPENDIX C

EXAMPLE 2

```

IMPLICIT REAL *8(A-H,Q-Z)
DIMENSION Y(2,513),YTST(513),M(8),E(513),F(513),TTST(8)
DATA M/3,7,15,31,63,127,255,511/
DATA TTST/.125,.25,.5,1.,2.,4.,8.,16./
RPI=3.141592653589793
QPI=RPI*RPI
IP1 = 1
IP2 = 2
IP3 = 3
IP4 = 4
IP5 = 5
IP6 = 6
IP7 = 7

```

```

EXB10000
EXB10100
EXB10200
EXB10300
EXB10400
EXB10500
EXB10600
EXB10700
EXB10800
EXB10900
EXB11000
EXB11100
EXB11200
EXB11300
EXB11400
EXB11500
EXB11600
EXB11700
EXB11800
EXB11900
EXB12000
EXB12100
EXB12200
EXB12300
EXB12400
EXB12500
EXB12600
EXB12700
EXB12800
EXB12900
EXB13000
EXB13100
EXB13200
EXB13300
EXB13400
EXB13500
EXB13600
EXB13700
EXB13800
EXB13900
EXB14000
EXB14100
EXB14200
EXB14300
EXB14400
EXB14500
EXB14600
EXB14700
EXB14800
EXB14900
EXB15000
EXB15100
EXB15200
EXB15300

```

MAIN LOOP FOR DIFFERENT STEP SIZES

```

C
C
C
DO 500 IA = 2,7
  RMAX = 0.0
  ID = 1
  KA=1
  KB=2
  T=0.0
  NA=M(IA)
  WRITE(6,903) NA
903  FORMAT('1',I30)
  NB=NA+1
  NUMM=0
  NUMD=0
  NC=NA+2
  HX=1./DFLOAT(NB)
  HT=HX
  ZPI = HX * RPI

C
C
C  INITILIZE
DO 30 IB = 1,NC
30  Y(KA,IB) = DEXP(-HX*DFLOAT(IB-1))*DSIN(ZPI*DFLOAT(IB-1))
  D1 = HT/HX
  A = .5DO*D1/HX
  B = 1.00 + 2.00 *A
  C = A
  A1 = A + D1
  A2 = 1.00-2.00*A+HT
  A3 = A-D1
  NUMM=NUMM+4
  NUMD = NUMD + 1
  E(1)=0.0
  F(1)=0.0

C
C
C  ONE STEP OF THE INTEGRATION - SOLVE SYSTEM OF EQUATIONS
50  Y(KA,1) = 0.0
  DO 100 IB = 2,NB
  CPT=B-C*E(IB-1)

```

	E(IB)=	A/CPT	EXB15400
	D =	A3 * Y(KA,IB-1) + A2*Y(KA,IB) + A1*Y(KA,IB+1)	EXB15500
100	F(IB)=(D+C*F(IB-1))/CPT		EXB15600
	Y(KB,NC)=0.0		EXB15700
	DO 200 IB=1,NA		EXB15800
	IC=NC-IB		EXB15900
200	Y(KB,IC)=E(IC)*Y(KB,IC+1)+F(IC)		EXB16000
	NUMM=NUMM+6*NA		EXB16100
	NUMD=NUMD+2*NA		EXB16200
	KA=3-KA		EXB16300
	KB=3-KB		EXB16400
	T=T+HT		EXB16500
	IF(T .LT. TTST(ID)) GO TO 50		EXB16600
C			EXB16700
C	OUTPUT SECTION		EXB16800
C			EXB16900
	WRITE (6,900) T,HX,A,B		EXB17000
	WRITE(6,904) Y(KA,IP1+1),Y(KA,IP2+1),Y(KA,IP3+1),Y(KA,IP4+1),		EXB17100
	1 Y(KA,IP5+1),Y(KA,IP6+1),Y(KA,IP7+1)		EXB17200
900	FORMAT(1X,5D26.15)		EXB17300
	WRITE(6,901) NUMM,NUMD		EXB17400
901	FORMAT(1X,3I30)		EXB17500
	WRITE(6,902)		EXB17600
902	FORMAT(' ')		EXB17700
904	FORMAT(1X,7D18.9)		EXB17800
	ID = ID + 1		EXB17900
	IF(T .LT. 16) GO TO 50		EXB18000
	IP1 = IP1 * 2		EXB18100
	IP2 = IP2 * 2		EXB18200
	IP3 = IP3 * 2		EXB18300
	IP4 = IP4 * 2		EXB18400
	IP5 = IP5 * 2		EXB18500
	IP6 = IP6 * 2		EXB18600
	IP7 = IP7 * 2		EXB18700
500	CONTINUE		EXB18800
	CALL EXIT		EXB18900
	END		EXB19000

APPENDIX D

EXAMPLE 3

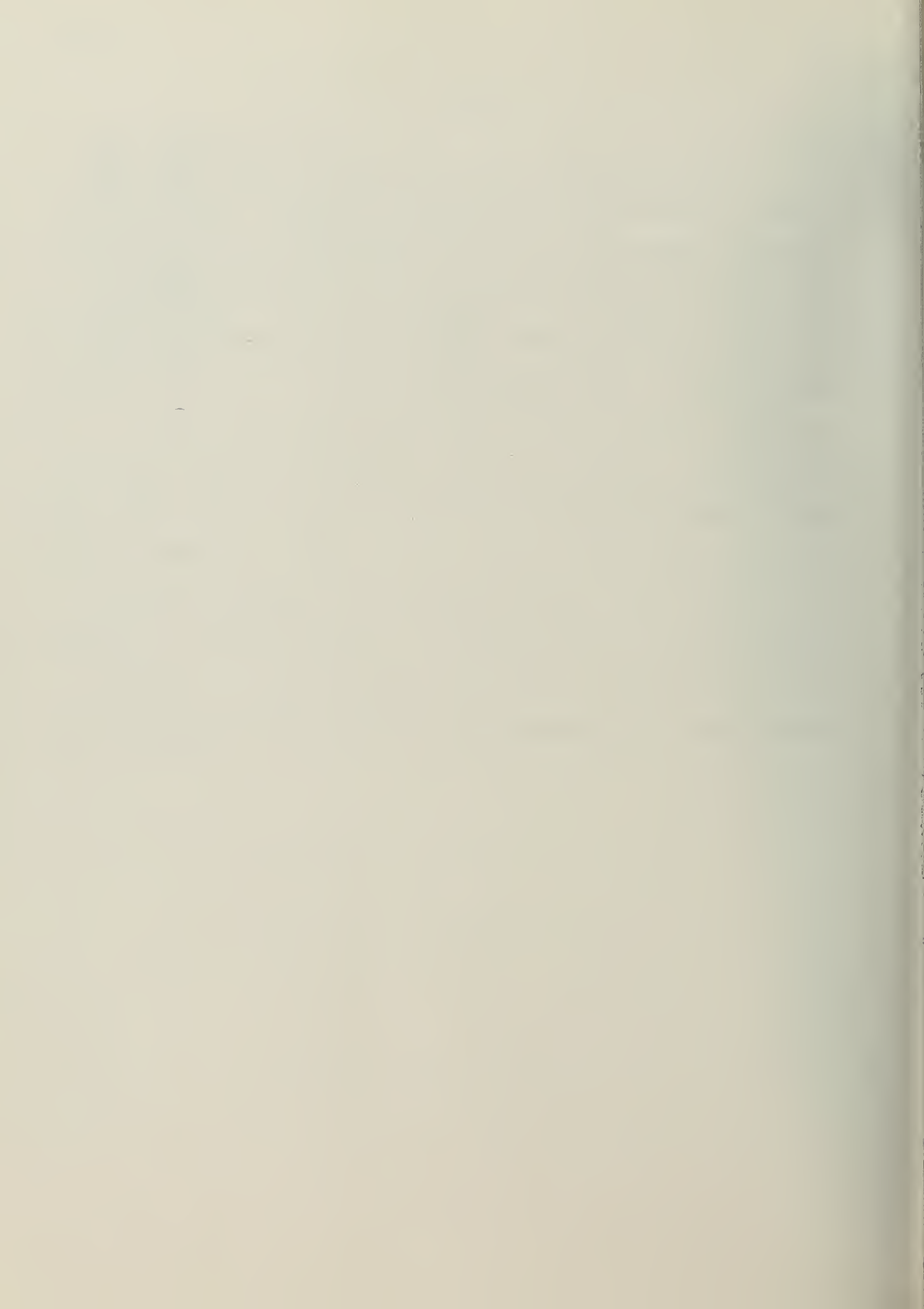
IMPLICIT REAL *8(A-H,O-Z)	EXC10000
DIMENSION TTST(8)	EXC10100
DIMENSION Y(2,1028),M(11),YTST(1028),RMAT(3,1028),B(1028)	EXC10200
DATA M/3,7,15,31,63,127,256,511,1023,2045,4095/	EXC10300
DATA TTST/.125,.25,.5,1.,2.,4.,8.,16./	EXC10400
CALL UNDERZ('OFF')	EXC10500
RK1 = 1.00	EXC10600
RK2 = 1.00	EXC10700
RPI=3.141592653589793	EXC10800
IP1 = 1	EXC10900
IP2 = 2	EXC11000
IP3 = 3	EXC11100
IP4 = 4	EXC11200
IP5 = 5	EXC11300
IP6 = 6	EXC11400
IP7 = 7	EXC11500
C	EXC11600
C	EXC11700
C	EXC11800
DO 500 IA = 2,7	EXC11900
RMAX = 0.0	EXC12000
ID = 1	EXC12100
KA=1	EXC12200
KB=2	EXC12300
T=0.0	EXC12400
NA=M(IA)	EXC12500
WRITE(6,903) NA	EXC12600
903 FORMAT('1',I30)	EXC12700
NB=NA+1	EXC12800
NUMM=0	EXC12900
NUMD=0	EXC13000
NC=NA+2	EXC13100
HX=1./DFLOAT(NB)	EXC13200
QPI = RPI*HX	EXC13300
HT=HX	EXC13400
C	EXC13500
C	EXC13600
C	EXC13700
DO 30 IB = 1,NC	EXC13800
V = DSIN(QPI*DFLOAT(IB-1))	EXC13900
30 Y(KA,IB) = V*V	EXC14000
C	EXC14100
C	EXC14200
C	EXC14300
SET UP VALUES FOR MATRIX	EXC14400
R = HT/(HX*HX)	EXC14500
B1 = 1.000 + HX	EXC14600
B2 = 1.000 + HX	EXC14700
TR = 2.00 * R	EXC14800
TRB1 = TR*B1	EXC14900
TRB2 = TR*B2	EXC15000
RMAT(1,1) = -R	EXC15100
RMAT(2,1) = 2.00 + TRB1	EXC15200
RMAT(3,1) = -TR	EXC15300
RMAT(1,NC) = -TR	

	RMAT(2,NC) = 2.DO + TRB2	EXC15400
	RMAT(3,NC) = -R	EXC15500
	RMAIN = 2.DO + TR	EXC15600
	RMANE = 2.DO -TR	EXC15700
	DO 10 I = 2,NB	EXC15800
	RMAT(1,I) = -R	EXC15900
	RMAT(2,I) = RMAIN	EXC16000
	RMAT(3,I) = -R	EXC16100
10	CONTINUE	EXC16200
C		EXC16300
C	DECOMPOSE THE MATRIX	EXC16400
C		EXC16500
	CALL DECOMP(NC,RMAT,J5,3)	EXC16600
	NUMM = 5+NA	EXC16700
	NUMD = 2+NA	EXC16800
	IF (J5 .LE. 0) GO TO 200	EXC16900
C		EXC17000
C	ONE STEP OF THE INTEGRATION - SOLVE SYSTEM OF EQUATIONS	EXC17100
C		EXC17200
50	B(1) = (2.DO - TRB1) *Y(KA,1) + TR *Y(KA,2)	EXC17300
	B(NC) = TR*Y(KA,NB) + (2.DO-TRB2)*Y(KA,NC)	EXC17400
C		EXC17500
C	SET UP THE RIGHT SIDE OF EQUATION	EXC17600
C		EXC17700
	DO 20 I = 2,NB	EXC17800
	B(I) = R*Y(KA,I-1) + RMANE*Y(KA,I) + R*Y(KA,I+1)	EXC17900
20	CONTINUE	EXC18000
	CALL SOLVE(NC,RMAT,B,Y,3,2,KB)	EXC18100
	NUMM = NUMM + 2*NC	EXC18200
	NUMD = NUMD + NC	EXC18300
	KA=3-KA	EXC18400
	KB=3-KB	EXC18500
	T=T+HT	EXC18600
	IF(T .LT. TTST(ID)) GO TO 50	EXC18700
C		EXC18800
C	OUTPUT SECTION	EXC18900
C		EXC19000
	DO 300 IB=1,NA	EXC19100
	YTST(IB) = Y(KA,IB+1) - Y(KB,IB+1)	EXC19200
	IF(RMAX.LT. DABS(YTST(IB))) RMAX = DABS(YTST(IB))	EXC19300
300	CONTINUE	EXC19400
	WRITE (6,900) T,HX	EXC19500
	WRITE(6,904) Y(KA,IP1+1),Y(KA,IP2+1),Y(KA,IP3+1),Y(KA,IP4+1),	EXC19600
1	Y(KA,IP5+1),Y(KA,IP6+1),Y(KA,IP7+1)	EXC19700
	WRITE(6,904) YTST(IP1),YTST(IP2),YTST(IP3),YTST(IP4),YTST(IP5),	EXC19800
1	YTST(IP6),YTST(IP7)	EXC19900
900	FORMAT(1X,5D26.15)	EXC20000
	WRITE(6,901) NUMM,NUMD	EXC20100
901	FORMAT(1X,3I30)	EXC20200
	WRITE(6,902)	EXC20300
902	FORMAT(' ')	EXC20400
904	FORMAT(1X,7D18.9)	EXC20500
	ID = ID + 1	EXC20600
	IF(T .LT. 16) GO TO 50	EXC20700
	IF(RMAX .LT. .001) CALL EXIT	EXC20800

	IP1 = IP1 * 2	EXC 20900
	IP2 = IP2 * 2	EXC 21000
	IP3 = IP3 * 2	EXC 21100
	IP4 = IP4 * 2	EXC 21200
	IP5 = IP5 * 2	EXC 21300
	IP6 = IP6 * 2	EXC 21400
	IP7 = IP7 * 2	EXC 21500
500	CONTINUE	EXC 21600
	CALL UNDERZ('ON')	EXC 21700
	CALL EXIT	EXC 21800
200	WRITE(6,2000)	EXC 21900
2000	FORMAT(' BAD MATRIX')	EXC 22000
	CALL EXIT	EXC 22100
	END	EXC 22200

VITA

Leonard Andrew Larsen was born on April 22, 1940. He attended elementary and secondary schools in Waukesha, Wisconsin. After graduating from Waukesha High School in 1958, he attended the University of Wisconsin, Madison, where the Bachelor of Science degree in Physics and Mathematics Education was conferred in 1962. Prior to attending the University of Illinois in 1967, he taught mathematics and physics at De Forest High School at De Forest, Wisconsin, for four years and at Conant High School in Hoffman Estates, Illinois, for one year. He received a Master of Science degree in Mathematics from the University of Illinois in August 1968. While continuing his education at the University of Illinois he taught at Illinois State University during the 1969 academic year. Beginning with the 1971 academic year, he has held the position of assistant professor of computer science at the University of Wisconsin, Eau Claire. He is currently an associate member of the Association for Computing Machinery and Sigma Xi.



U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

C00-1469-0212

2. TITLE

AUTOMATIC SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS

TYPE OF DOCUMENT (Check one):

- ☐ a. Scientific and technical report
☐ b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

- ☒ c. Other (Specify) Thesis

RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.
☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
☐ c. Make no announcement or distribution.

REASON FOR RECOMMENDED RESTRICTIONS:

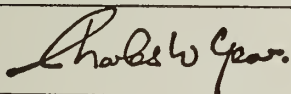
SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear, Professor
and Principal Investigator

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature



Date

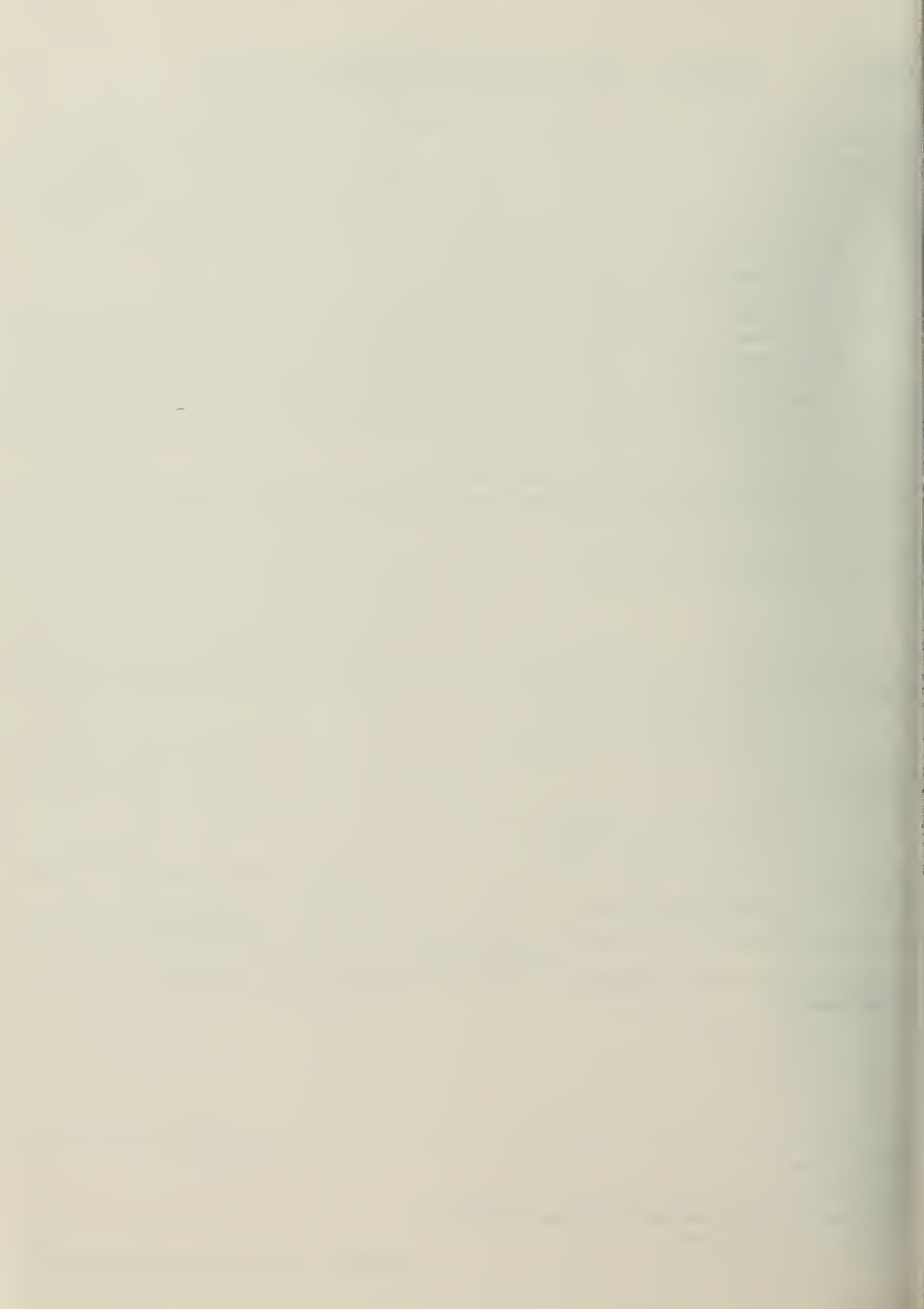
October 1972

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION
RECOMMENDATION:

PATENT CLEARANCE:

- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.
☐ b. Report has been sent to responsible AEC patent group for clearance.
☐ c. Patent clearance not required.



BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-72-546	2.	3. Recipient's Accession No.
4. Title and Subtitle AUTOMATIC SOLUTIONS OF PARTIAL DIFFERENTIAL EQUATIONS		5. Report Date October 1972	
7. Author(s) Leonard Andrew Larsen		6.	
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801		8. Performing Organization Rept. No.	
12. Sponsoring Organization Name and Address US AEC Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois		10. Project/Task/Work Unit No. 1469-Gear	
		11. Contract/Grant No. US AEC AT(11-1)1469	
15. Supplementary Notes		13. Type of Report & Period Covered Thesis (Ph.D.)	
		14.	
16. Abstracts The problem of the present paper is to look for an automatic method that can be applied to some subset of partial differential equations. The primary value of a program involving an automatic method would be to provide the person who needs only a few runs with a particular type of equation the opportunity to find a solution, within acceptable tolerances, without having to write and debug a specialized program.			
17. Key Words and Document Analysis. 17a. Descriptors partial differential equations automatic method spatial discretization			
7b. Identifiers/Open-Ended Terms			
7c. COSATI Field/Group			
8. Availability Statement unlimited distribution		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 129
		20. Security Class (This Page) UNCLASSIFIED	22. Price



UNIVERSITY OF ILLINOIS-URBANA



3 0112 054457137